Report

v. 2.0

Customer

Lemma Labs

Smart Contract Audit

InfinityPools

5th September 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog **11**

2 Introduction **12**

3 Project scope **13**

4 Methodology **15**

5 Our findings **16**

6 Major Issues **17**
CVF-1. INFO . . . 17
CVF-2. INFO . . . 18
CVF-3. FIXED . . . 19
CVF-4. FIXED . . . 20
CVF-17. FIXED . . . 21
CVF-19. FIXED . . . 22
CVF-20. FIXED . . . 22
CVF-24. INFO . . . 23

7 Moderate Issues **24**
CVF-5. INFO . . . 24
CVF-7. FIXED . . . 24
CVF-8. INFO . . . 25
CVF-9. INFO . . . 26
CVF-10. INFO . . . 26
CVF-11. INFO . . . 28
CVF-12. INFO . . . 29
CVF-13. INFO . . . 30
CVF-14. INFO . . . 30
CVF-15. INFO . . . 31
CVF-16. INFO . . . 32
CVF-25. FIXED . . . 32
CVF-26. INFO . . . 33
CVF-27. INFO . . . 33
CVF-28. INFO . . . 34
CVF-29. INFO . . . 35
CVF-30. FIXED . . . 35
CVF-31. INFO . . . 36
CVF-32. INFO . . . 37
CVF-33. INFO . . . 38
CVF-34. INFO . . . 39
CVF-35. INFO . . . 39
CVF-36. INFO . . . 40

CVF-37. FIXED 41
CVF-38. INFO 41
CVF-39. INFO 41
CVF-40. INFO 42
CVF-41. INFO 42
CVF-42. INFO 43
CVF-43. INFO 43
CVF-44. INFO 44
CVF-45. INFO 44
CVF-46. INFO 45
CVF-47. INFO 45
CVF-48. INFO 46
CVF-49. FIXED 46
CVF-50. FIXED 47
CVF-51. INFO 47
CVF-52. INFO 48
CVF-53. INFO 49
CVF-54. INFO 49
CVF-55. INFO 50
CVF-56. INFO 50
CVF-57. INFO 51
CVF-58. INFO 52
CVF-59. INFO 52
CVF-60. FIXED 52

8 Minor Issues **53**
CVF-61. FIXED 53
CVF-62. INFO 53
CVF-63. INFO 53
CVF-64. FIXED 54
CVF-65. FIXED 54
CVF-66. FIXED 55
CVF-67. FIXED 55
CVF-68. INFO 55
CVF-69. INFO 56
CVF-70. INFO 56
CVF-71. INFO 57
CVF-72. INFO 57
CVF-73. INFO 58
CVF-74. INFO 58
CVF-75. INFO 59
CVF-76. INFO 59
CVF-77. INFO 60
CVF-78. INFO 60
CVF-79. INFO 61
CVF-80. INFO 61

CVF-81. INFO 61
CVF-82. INFO 62
CVF-83. INFO 62
CVF-84. INFO 62
CVF-85. INFO 62
CVF-86. INFO 63
CVF-87. INFO 63
CVF-88. INFO 63
CVF-89. INFO 63
CVF-90. INFO 64
CVF-91. INFO 64
CVF-92. INFO 64
CVF-93. INFO 64
CVF-94. INFO 65
CVF-95. INFO 65
CVF-96. INFO 65
CVF-97. INFO 65
CVF-98. INFO 66
CVF-99. INFO 66
CVF-100. INFO 66
CVF-101. INFO 66
CVF-102. INFO 67
CVF-103. INFO 67
CVF-104. INFO 67
CVF-105. INFO 67
CVF-106. INFO 68
CVF-107. INFO 68
CVF-108. INFO 68
CVF-109. INFO 69
CVF-110. INFO 69
CVF-111. INFO 69
CVF-112. INFO 69
CVF-113. INFO 70
CVF-114. INFO 70
CVF-115. INFO 70
CVF-116. INFO 71
CVF-117. INFO 71
CVF-118. INFO 71
CVF-119. INFO 72
CVF-120. INFO 72
CVF-121. INFO 72
CVF-122. INFO 73
CVF-123. INFO 73
CVF-124. INFO 73
CVF-125. INFO 73
CVF-126. INFO 74

CVF-127. INFO 75
CVF-128. INFO 76
CVF-129. INFO 76
CVF-130. INFO 77
CVF-131. INFO 77
CVF-132. INFO 77
CVF-133. INFO 77
CVF-134. INFO 78
CVF-135. INFO 78
CVF-136. INFO 78
CVF-137. INFO 79
CVF-138. INFO 79
CVF-139. INFO 79
CVF-140. INFO 80
CVF-141. INFO 80
CVF-142. INFO 81
CVF-143. INFO 81
CVF-144. INFO 82
CVF-145. INFO 82
CVF-146. INFO 82
CVF-147. INFO 83
CVF-148. INFO 83
CVF-149. INFO 83
CVF-150. INFO 84
CVF-151. INFO 84
CVF-152. INFO 84
CVF-153. INFO 85
CVF-154. INFO 85
CVF-155. INFO 85
CVF-156. INFO 85
CVF-157. INFO 86
CVF-158. INFO 86
CVF-159. INFO 87
CVF-160. INFO 88
CVF-161. INFO 89
CVF-162. INFO 89
CVF-163. INFO 89
CVF-164. INFO 90
CVF-165. INFO 91
CVF-166. INFO 91
CVF-167. INFO 92
CVF-168. INFO 92
CVF-169. INFO 93
CVF-170. INFO 93
CVF-171. INFO 94
CVF-172. INFO 95

CVF-173. INFO 95
CVF-174. INFO 96
CVF-175. INFO 96
CVF-176. INFO 97
CVF-177. INFO 97
CVF-178. INFO 98
CVF-179. INFO 99
CVF-180. INFO 100
CVF-181. INFO 101
CVF-182. INFO 102
CVF-183. INFO 102
CVF-184. INFO 102
CVF-185. INFO 103
CVF-186. INFO 103
CVF-187. INFO 103
CVF-188. INFO 104
CVF-189. INFO 104
CVF-190. INFO 104
CVF-191. INFO 105
CVF-192. INFO 105
CVF-193. INFO 105
CVF-194. INFO 106
CVF-195. INFO 106
CVF-196. INFO 106
CVF-197. INFO 107
CVF-198. INFO 107
CVF-199. INFO 107
CVF-200. INFO 108
CVF-201. INFO 108
CVF-202. INFO 108
CVF-203. INFO 109
CVF-204. INFO 109
CVF-205. INFO 109
CVF-206. INFO 110
CVF-207. INFO 110
CVF-208. INFO 110
CVF-209. INFO 111
CVF-210. INFO 111
CVF-211. INFO 111
CVF-212. INFO 112
CVF-213. INFO 112
CVF-214. INFO 112
CVF-215. INFO 113
CVF-216. INFO 113
CVF-217. INFO 113
CVF-218. INFO 114

CVF-219. INFO 114
CVF-220. INFO 114
CVF-221. INFO 115
CVF-222. INFO 115
CVF-223. INFO 115
CVF-224. INFO 115
CVF-225. INFO 116
CVF-226. INFO 116
CVF-227. INFO 116
CVF-228. INFO 116
CVF-229. INFO 117
CVF-230. INFO 117
CVF-231. INFO 117
CVF-232. INFO 118
CVF-233. INFO 118
CVF-234. INFO 118
CVF-235. INFO 119
CVF-236. INFO 119
CVF-237. INFO 120
CVF-238. INFO 120
CVF-239. INFO 121
CVF-240. INFO 121
CVF-241. INFO 121
CVF-242. INFO 122
CVF-243. INFO 122
CVF-244. INFO 122
CVF-245. INFO 123
CVF-246. INFO 123
CVF-247. INFO 123
CVF-248. INFO 124
CVF-249. FIXED 124
CVF-250. INFO 124
CVF-251. INFO 125
CVF-252. INFO 125
CVF-253. INFO 125
CVF-254. INFO 126
CVF-255. INFO 126
CVF-256. INFO 126
CVF-257. FIXED 127
CVF-258. FIXED 127
CVF-259. INFO 127
CVF-260. INFO 128
CVF-261. INFO 128
CVF-262. INFO 128
CVF-263. INFO 128
CVF-264. INFO 129

CVF-265. INFO 129
CVF-266. INFO 130
CVF-267. INFO 130
CVF-268. INFO 131
CVF-269. INFO 131
CVF-270. INFO 131
CVF-271. INFO 132
CVF-272. INFO 132
CVF-273. INFO 132
CVF-274. INFO 133
CVF-275. INFO 133
CVF-276. INFO 133
CVF-277. INFO 133
CVF-278. INFO 134
CVF-279. INFO 134
CVF-280. INFO 134
CVF-281. INFO 134
CVF-282. INFO 135
CVF-283. INFO 135
CVF-284. INFO 135
CVF-285. INFO 135
CVF-286. INFO 136
CVF-287. INFO 136
CVF-288. INFO 136
CVF-289. INFO 136
CVF-290. INFO 137
CVF-291. FIXED 137
CVF-292. INFO 137
CVF-293. INFO 138
CVF-294. INFO 138
CVF-295. INFO 139
CVF-296. INFO 139
CVF-297. INFO 139
CVF-298. INFO 140
CVF-299. INFO 140
CVF-300. INFO 140
CVF-301. INFO 141
CVF-302. FIXED 141
CVF-303. INFO 142
CVF-304. INFO 142
CVF-305. INFO 143
CVF-306. INFO 143
CVF-307. FIXED 144
CVF-308. FIXED 144
CVF-309. INFO 145
CVF-310. INFO 145

CVF-311. INFO 145
CVF-312. INFO 146
CVF-313. INFO 146
CVF-314. INFO 146
CVF-315. INFO 147
CVF-316. INFO 147
CVF-317. INFO 147
CVF-318. INFO 148
CVF-319. INFO 148
CVF-320. INFO 148
CVF-321. INFO 149
CVF-322. FIXED 149
CVF-323. FIXED 149
CVF-324. FIXED 150
CVF-325. FIXED 150
CVF-326. FIXED 150
CVF-327. INFO 151
CVF-328. FIXED 151
CVF-329. FIXED 151
CVF-330. FIXED 152
CVF-331. FIXED 152
CVF-332. FIXED 152
CVF-333. FIXED 153
CVF-334. INFO 153
CVF-335. INFO 153
CVF-336. INFO 153
CVF-337. INFO 154
CVF-338. INFO 154
CVF-339. INFO 154
CVF-340. FIXED 155
CVF-341. FIXED 155
CVF-342. FIXED 155
CVF-343. INFO 156
CVF-344. INFO 156
CVF-345. FIXED 156
CVF-346. INFO 157
CVF-347. INFO 157
CVF-348. INFO 157
CVF-349. INFO 157
CVF-350. INFO 158
CVF-351. INFO 158
CVF-352. INFO 158
CVF-353. INFO 158
CVF-354. INFO 159
CVF-355. INFO 159
CVF-356. INFO 159

CVF-357. INFO 159
CVF-358. INFO 160
CVF-359. INFO 160
CVF-360. INFO 160
CVF-361. INFO 160
CVF-362. FIXED 161
CVF-363. FIXED 161
CVF-364. INFO 161
CVF-365. INFO 162
CVF-366. INFO 162
CVF-367. INFO 162
CVF-368. INFO 163
CVF-369. INFO 163
CVF-370. INFO 163
CVF-371. INFO 164
CVF-372. INFO 164
CVF-373. INFO 164
CVF-374. INFO 165
CVF-375. INFO 165
CVF-376. INFO 165
CVF-377. INFO 165
CVF-378. INFO 166
CVF-379. INFO 166
CVF-380. INFO 166
CVF-381. INFO 166
CVF-382. INFO 167
CVF-383. INFO 167
CVF-384. INFO 167
CVF-385. INFO 167
CVF-386. INFO 168
CVF-387. INFO 168
CVF-388. INFO 168
CVF-389. INFO 168
CVF-390. INFO 169
CVF-391. INFO 169
CVF-392. INFO 169
CVF-393. INFO 169
CVF-394. INFO 170
CVF-395. INFO 170

1 Changelog

#	Date	Author	Description
0.1	26.08.24	A. Zveryanskaya	Initial Draft
0.2	26.08.24	A. Zveryanskaya	Minor revision
1.0	27.08.24	A. Zveryanskaya	Release
1.1	5.09.24	A. Zveryanskaya	CVF-1, 4 typo fixed
2.0	5.09.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Infinity Pools is a decentralized exchange that offers unlimited leverage on any asset, with no liquidations, no counterparty risk, and no oracles.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/		
Constants.sol	InfinityPool.sol	InfinityPoolDeployer.sol
InfinityPoolFactory.sol	InfinityPoolState.sol	PoolReader.sol
interfaces/		
IInfinityPool.sol	IInfinityPoolDeployer.sol	IInfinityPoolFactory.sol
IInfinityPool LoanState.sol	IInfinityPoolPayment Callback.sol	IInfinityPoolState.sol
libraries/external/		
Advance.sol	LP.sol	NewLoan.sol
Spot.sol	Structs.sol	Swapper.sol
libraries/helpers/		
DeadlineHelper.sol	PoolHelper.sol	
types/Optional/		
OptInt256.sol		
libraries/internal/		
BoxcarTubFrame.sol	BucketRolling.sol	BytesLib.sol
Capper.sol	DailyJumps.sol	DeadlineFlag.sol
DeadlineJumps.sol	DropFaberTotals.sol	EachPayoff.sol
EraBoxcarMidSum.sol	EraFaberTotals.sol	Fees.sol

libraries/internal/		
GapStagedFrame.sol	GrowthSplitFrame.sol	JumpyAnchorFaber.sol
JumpyFallback.sol	Payoff.sol	SparseFloat.sol
UserPay.sol	Utils.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 8 major, and a few less important issues.

Fixed 5 out of 8 issues

6 Major Issues

CVF-1 INFO

- **Category** Flaw
- **Source** SparseFloat.sol

Description The `FloatBits` library incorrectly handles left shifts (<<) and overflow checks for negative significands within the `truncate` function. Specifically:

- Left shifting a negative signed integer can lead to undefined behavior due to the movement of the sign bit.
- The current overflow check is unreliable for negative significands. Left shifting a negative value can change the sign bit, potentially leading to incorrect overflow detection. In this scenario, when a large positive shift is applied to a negative significand, the resulting value might become very large (due to the sign bit moving left), potentially exceeding the maximum value representable by an `int128`. However, the flawed overflow check might not detect this, leading to incorrect results.

Recommendation We recommend modifying the code to correctly handle left shifts and overflow checks for negative significands:

1. Use the signed left shift operator << and cast the `shift` amount to an unsigned integer type `uint128` to ensure correct behavior for both positive and negative significands.
2. After performing the left shift, check for overflow by right-shifting the result by the same amount and comparing it to the original value. This ensures that the sign bit is preserved and the overflow is detected accurately.

Replace the following code:

```
24:     function truncate(Info memory self, int128 newExponent)
   ↪ internal pure returns (int128 resultSignificand) {
25:         int128 shift = self.exponent - newExponent;
26:         if (shift <= 0) {
...
29:         } else {
30:             //the shift cannot be greater than 256 or this(
   ↪ uint256(256-shift)) fails silently with underflow
**31:             if (!(int128ShiftRightUnsigned(self.significand, ((
   ↪ WORD_SIZE - shift) > 0 ? uint128(WORD_SIZE - shift) : 0)) ==
   ↪ 0)) {
32:                 revert("FloatBits: Truncate overflow");
33:             }
34:             resultSignificand = self.significand << uint128(shift
   ↪ );**
35:         }
```

with:

```
24:     function truncate(Info memory self, int128 newExponent)
   ↪ internal pure returns (int128 resultSignificand) {
25:         int128 shift = self.exponent - newExponent;
26:         if (shift <= 0) {
...
29:         } else {
30:             //the shift cannot be greater than 256 or this(
   ↪ uint256(256-shift)) fails silently with underflow
**31:             resultSignificand = self.significand << uint128(
   ↪ shift);
32:             require(resultSignificand >> uint128(shift) == self.
   ↪ significand, "FloatBits: Truncate overflow");**
33:         }
```

Client Comment This code works correctly and as intended. A negative significand input (self.significand) here will revert (as shift >= 1 in this context). A negative significand output (resultSignificand) is correct behaviour, so does not and should not revert. The FloatBits type holds a run of 128 bits from a longer decimal expansion - more significant bits from the expansion can vary, and are not related to the 128th bit held (ie. sign bit).

CVF-2 INFO

- **Category** Flaw
- **Source** SparseFloat.sol

Description The `repack` function of the `QuadPacker` library incorrectly calculates the position of the most significant bit (MSB) for negative significands in the `repack` function of the `QuadPacker` library. The vulnerable line of code:

```
83:     int128 intHighestBit = int128(int256(mostSignificantBit(
   ↪ uint256(uint128(bits.significand)))));
```

This calculation casts the signed integer `bits.significand` to `uint256` before passing it to the `mostSignificantBit` function. This casting changes the binary representation of negative numbers, leading to incorrect MSB calculations.

Recommendation Modify the code to correctly handle negative significands when calculating the most significant bit. One approach is to use a conditional statement to handle negative and positive significands separately.

Client Comment This code works correctly and as intended. The highest set bit of a negative int128 is at position 127, which is precisely what intHighestBit will be set to on this line. This line of code is therefore correct individually, and its behaviour is indeed what is required for the enclosing 'repack' function to work correctly.

CVF-3 FIXED

- **Category** Flaw
- **Source** SparseFloat.sol

Description The current `truncate` function of the `FloatBits` library incorrectly uses an unsigned right shift `int128ShiftRightUnsigned` on a signed integer `self.significand` when the shift amount `-shift` is negative. This is problematic because right shifting a negative signed integer should preserve the sign bit (the leftmost bit), effectively filling in the leftmost bits with 1s. Furthermore, the unsigned right shifts fill in the leftmost bits with 0s, regardless of the original sign. In this specific scenario, when `shift` is negative, the code intends to perform a left shift (to increase the value of the significand). However, due to the incorrect use of an unsigned right shift, the sign bit is not preserved, leading to incorrect results for negative significands.

Recommendation Replace the `int128ShiftRightUnsigned` function with a direct signed right shift >> operation. For instance, replace:

```
24:     function truncate(Info memory self, int128 newExponent)
   ↪ internal pure returns (int128 resultSignificand) {
25:         int128 shift = self.exponent - newExponent;
26:         if (shift <= 0) {
27:             if (-shift < WORD_SIZE) **resultSignificand =
   ↪ int128ShiftRightUnsigned(self.significand, uint128(-shift));**
28:             else resultSignificand = 0;
...
```

with:

```
24:     function truncate(Info memory self, int128 newExponent)
   ↪ internal pure returns (int128 resultSignificand) {
25:         int128 shift = self.exponent - newExponent;
26:         if (shift <= 0) {
27:             if (-shift < WORD_SIZE) **resultSignificand = self.
   ↪ significand >> uint128(-shift);**
28:             else resultSignificand = 0;
...
```

This change will ensure that negative significands are shifted correctly, preserving the sign bit and producing the expected results for left shifts.

Client Comment This code works correctly and as intended. The int128ShiftRightUnsigned is intended, as the enclosing 'truncate' function shifts bringing in zeros from the right. This is the correct behaviour given bits to the right are known to be zero, such as on the output of 'unpack'. Since the only calls to 'truncate' which are needed, are on an output of 'unpack', 'truncate' has been refactored to enforce this, making correctness more obvious.

CVF-4 FIXED

- **Category** Overflow/Underflow
- **Source** Spot.sol

Description The `flush` function of the `Spot` library attempts to convert the result of `self.tickBin + 1` and `self.tickBin - 1` to `int24` data types. However, it does not include any checks to ensure that the results of these operations fall within the valid range of an `int24`. In case `self.tickBin` is already at its maximum or minimum value, adding or subtracting 1 could lead to an overflow or underflow, respectively.

```
27:     function flush(PoolState storage self, bool isUp) internal {
28:         if (isUp) self.joinStaged.flush(self, int256(**int24(self
    ↪ .tickBin + 1)**));
29:         else self.joinStaged.flush(self, int256(**int24(self.
    ↪ tickBin - 1)**));
30:     }
```

An overflow occurs when the result of an arithmetic operation exceeds the maximum value representable by the data type, while an underflow occurs when the result falls below the minimum representable value. In this case, an overflow would result in an unexpectedly small negative value, and an underflow would result in an unexpectedly large positive value.

Recommendation To ensure that type conversions cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[1] for casting operations where possible. If overflows and underflows are possible and desired here, explicitly document and provide guidance on how to use the function correctly.

Client Comment No repro. of over/under-flow has been provided - indeed none is possible, as this code is called only from either `_uptick` or `_downtick`, where a range check has already been performed. The conversion to int24 has been removed, and these lines have also been folded into `_uptick` and `_downtick`, to make the correctness more obvious.

[1] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

CVF-17 FIXED

- **Category** Unclear behavior
- **Source** GapStagedFrame.sol

Description The `meet` function in the Gap library is designed to update the `start` and `stop` values of a `Gap.Info` struct based on another `Gap.Info` struct. However, the current implementation does not prevent a situation where `self.start` could become greater than `self.stop`.

```
23:     function meet(Info storage self, Info memory other) internal
    ↪ {
24:         self.start = SignedMath.max(self.start, other.start);
25:         self.stop = SignedMath.min(self.stop, other.stop);
26:     }
```

If `other.start` is greater than `stop`, the first line will update `self.start` to be greater than `self.stop`, leading to an inconsistent state where the gap's start is after its end.

Recommendation Consider one of the following options:

1. Explicitly forbid inconsistent state: As a last line of the function, add a check to ensure that `self.start < self.stop`. If the condition is not met, the function could revert the transaction or take other appropriate action.
2. Document the behavior: If there is a valid reason for allowing `self.start` to be greater than `self.stop` in certain scenarios, this should be clearly documented in the code comments. The documentation should explain the rationale behind this behavior and provide guidance to users on how to interpret and handle such cases.

Client Comment No repro. has been provided - indeed none is possible, for the following reason. The expected ordering self.start <= self.stop is guaranteed by the fact that every Gap object satisfies self.start <= tickBin < self.stop. An assertion has been added where preservation of this property was not self-evident.

CVF-19 FIXED

- **Category** Unclear behavior
- **Source** OptInt256.sol

Description The `wrap` function in the `OptInt256` library is designed to convert a standard `int256` value into an `OptInt256` instance. However, it incorrectly handles the special value `type(int256).min`, which is intended to represent an undefined or "none" value in the `OptInt256` type.

```
27:     function wrap(int256 x) pure returns (OptInt256 result) {
28:         result = OptInt256.wrap(x);
29:     }
```

The current implementation directly wraps `type(int256).min` into an `OptInt256` instance, making it indistinguishable from a valid, defined `OptInt256` value. This can lead to unexpected behavior and errors in code that relies on the distinction between defined and undefined `OptInt256` values.

Recommendation Consider reverting the `wrap` function when the input value is `type(int256).min`, for instance:

```
if (x == type(int256).min) revert("Cannot␣wrap␣type(int256).min");
```

This will prevent the incorrect wrapping of this special value and signal an error to the caller, indicating that the input cannot be represented as a valid `OptInt256` instance.

Client Comment This code works correctly and as intended. It is the expected behaviour for 'wrap' to accept all values, including the special value representing an empty optional. To make this property clearer, the 'unwrap' function has been renamed to 'get'.

CVF-20 FIXED

- **Category** Unclear behavior
- **Source** SparseFloat.sol

Description The `add` function within the `FloatBits` library is designed to add two floating-point numbers represented by their `Info` structs. However, the addition operation implemented in this function is not commutative. This means that the order of the operands `self` and `other` affects the result of the addition and could lead to confusion or errors.
In most mathematical contexts, addition is commutative (e.g., a + b = b + a). However, in the context of floating-point arithmetic with limited precision, the order of operations can sometimes lead to slight differences in results due to rounding errors.

Recommendation Consider implementing one of the two potential ways to avoid potential errors or unexpected results:

1. Make the `add` function commutative: Modify the implementation of the `add` function to ensure that the order of operands does not affect the result.

2. Clearly document the non-commutative behavior: If there is a specific reason why the add function cannot be made commutative, this should be clearly documented to provide guidance on how to use the function correctly.

```
17: function add(Info memory self, Info memory other) internal pure
    ↪ returns (Info memory result) {
```

Client Comment This code works correctly and as intended. As a result of the refactoring from CVF-3, the function signature of `add` is now `add(Info, Quad)`, for which it is clearer commutativity is not expected.

CVF-24 INFO

- **Category** Overflow/Underflow
- **Source** PoolHelper.sol

Description The code performs several conversions to `uint256` without proper checks for underflow. Underflow occurs when a negative signed integer is converted to an unsigned integer, resulting in a large positive value. This can lead to unexpected behavior and incorrect calculations in the subsequent operations.

Recommendation To ensure that type casts cannot underflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[2] for casting operations where possible.

```
28: return LOG1PPC / fromUint256(1 << uint256(subSplits(splits)));
```

```
36: return int256(1 << uint256(splits));
```

```
68: return BoxcarTubFrame.apply_(self, bin >> uint256(subSplits(
    ↪ splits)));
```

```
118: return (bin - BINS(splits) / 2) << (TICK_SUB_SLITS - uint256(
    ↪ subSplits(splits)));
```

```
162: return LOG1PPC / fromUint256(1 << TICK_SUB_SLITS);
```

Client Comment No repro. of underflow has been provided - indeed none is possible, for the following reasons. The 'splits' function argument used on these lines is always pool.splits (an argument is required to provide this pool state into helper functions). Therefore all subSplits calls return pool.splits - 12. As pool.splits >= 12 is enforced at pool creation (line 32 of InfinityPoolFactory.createPool), and pool.splits does not change thereafter, no underflow is possible. Note as well, libraries are used in our codebase to satisfy the EVM contract size limit, and do not indicate encapsulation.

[2] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

7 Moderate Issues

CVF-5 INFO

- **Category** Overflow/Underflow
- **Source** PoolHelper.sol

Description The `BINS` function of `PoolHelper` performs a conversion to `int256` without proper checks for overflow:

```
36:     return int256(1 << uint256(splits));
```

Overflow occurs when a value exceeds the maximum limit of the data type, leading to unexpected behavior and incorrect calculations in subsequent operations.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[3] for casting operations. If overflows and underflows are possible and desired here, explicitly document and provide guidance on how to use the function correctly.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reason. The 'splits' function argument used on this line is always pool.splits (an argument is required to provide this pool state to the function). As pool.splits <= 19 is enforced at pool creation (line 32 of InfinityPoolFactory.createPool), and pool.splits does not change thereafter, no overflow is possible. Note as well, libraries are used in our codebase to satisfy the EVM contract size limit, and do not indicate encapsulation.

CVF-7 FIXED

- **Category** Overflow/Underflow
- **Source** BoxcarTubFrame.sol

```
32: if (((offset < 0) || (offset >= int256(1 << scale.toUint256())))
   ↪ ) revert OffsetOutOfRange();
```

```
34: int256 node = (int256(1 << scale.toUint256()) + offset);
```

```
55: int256 length = int256(1 << scale.toUint256());
```

Description Many instances of the codebase include the left shift and type conversion logic implemented as `int256(1 << scale.toUint256())` or similarly. It performs left shift operations << on integer values and then converts the results to `int256`. This can lead to integer overflows in the following scenarios:

[3] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

1. If the result of the left shift exceeds the maximum value representable by the integer type, an overflow occurs. This can lead to unexpected and incorrect values.
2. When converting the result of the left shift (which could already be overflowing) to `int256`, further overflow can occur if the value exceeds the maximum limit of `int256`.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast`library[4] for casting operations and using safe operations where possible.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reason. Each of the functions `apply_`, addRange, and add are only called through a helper function (of the same name) which passes 12 as `scale` argument. Therefore the value of `scale` is 12 in all cases here, and there can be no overflow. The `apply_` and addRange functions have been refactored to make this property more obvious.

CVF-8 INFO

- **Category** Overflow/Underflow
- **Source** JumpyFallback.sol

```
39: int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

```
78: int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

```
117: int256 cursor = ((1 << splits.toUint256()).toInt256() + offset)
   ↪ ;
```

Description There is many instances across the codebase performing left shift operations << on integer values without proper checks for potential overflows. If the result of a left shift exceeds the maximum value representable by the integer type, an overflow occurs, leading to unexpected and incorrect values.

Recommendation Consider mitigating the risk of overflows in left shift operations by using safe operations and/or implementing proper overflow checks.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reason. The 'splits' function argument used on these lines is always pool.splits (an argument is required to provide this pool state into helper functions). As pool.splits <= 19 is enforced at pool creation (line 32 of InfinityPoolFactory.createPool), and pool.splits does not change thereafter, no overflow is possible. Note as well, libraries are used in our codebase to satisfy the EVM contract size limit, and do not indicate encapsulation.

[4] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

CVF-9 INFO

- **Category** Overflow/Underflow
- **Source** GrowthSplitFrame.sol

```
36: self.activeLeaf = int256(1 << pool.splits.toUint256()) + initBin
   ↪ ;
```

```
68: int256 start = (int256(1 << scale.toUint256()) + startIdx);
int256 stop = (int256(1 << scale.toUint256()) + stopIdx);
```

Description Many instances of the codebase include the left shift and type conversion logic implemented as `int256(1 << scale.toUint256())` or similarly. It performs left shift operations << on integer values and then converts the results to `int256`. This can lead to integer overflows in the following scenarios:

1. If the result of the left shift exceeds the maximum value representable by the integer type, an overflow occurs. This can lead to unexpected and incorrect values.
2. When converting the result of the left shift (which could already be overflowing) to `int256`, further overflow can occur if the value exceeds the maximum limit of `int256`.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[5] for casting operations and using safe operations where possible.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reasons. On line 36, there can be no overflow because pool.splits is at most 19 (enforced line 32 of InfinityPoolFactory.createPool). On lines 68-69, there can be no overflow because the value of 'scale' has already been checked to be at most 12.

CVF-10 INFO

- **Category** Overflow/Underflow
- **Source** GrowthSplitFrame.sol

```
112: int256 node = (int256(1 << scale.toUint256()) + offset);
```

```
127: Quad nextCoef = smallAtTailReader(self, (int256(1 << depth.
   ↪ toUint256()) + int256(offset >> (scale - depth).toUint256())))
   ↪ ;
    bool rightChild = ((offset & int256(1 << ((scale - depth).
       ↪ toUint256() - 1))) != 0);
```

```
137: int256 node = (int256(1 << scale.toUint256()) + offset);
```

[5] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

```
151: if (((stopIdx % int256(1 << (scale - int256(MIN_SPLITS)).
   ↪ toUint256())) == 0)) {
```

```
154: if (!(scale >= int256(MIN_SPLITS) && (((stopIdx & int256(-1 <<
   ↪ scale.toUint256())) == 0)))) revert InvalidSubLeftSumArguments
   ↪ ();
```

```
156: self, 0, POSITIVE_ZERO, int256((1 << MIN_SPLITS) + (stopIdx >>
   ↪ (scale - int256(MIN_SPLITS)).toUint256()).toUint256())
```

```
174: if (((stopIdx % int256(1 << (scale - int256(MIN_SPLITS)).
   ↪ toUint256())) == 0)) {
```

```
179: self, pool, 0, POSITIVE_ZERO, int256((1 << MIN_SPLITS) + (
   ↪ stopIdx >> (scale - int256(MIN_SPLITS)).toUint256()).toUint256
   ↪ ())
```

```
184: Quad nextCoef = smallAtNowReader(self, pool, int256((1 << depth
   ↪ .toUint256()) + uint256(stopIdx >> (scale - depth).toUint256()
   ↪ )));
```

```
186: bool rightChild = ((stopIdx & int256(1 << ((scale - depth) - 1)
   ↪ .toUint256())) != 0);
```

```
200: if (!((0 <= startIdx) && (stopIdx <= int256(1 << scale.
   ↪ toUint256())))) revert InvalidSumRangeArguments();
```

Description Many instances of the codebase include the left shift and type conversion logic implemented as `int256(1 << scale.toUint256())` or similarly. It performs left shift operations << on integer values and then converts the results to `int256`. This can lead to integer overflows in the following scenarios:

1. If the result of the left shift exceeds the maximum value representable by the integer type, an overflow occurs. This can lead to unexpected and incorrect values.
2. When converting the result of the left shift (which could already be overflowing) to `int256`, further overflow can occur if the value exceeds the maximum limit of `int256`.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[6] for casting operations and using safe operations where possible.

[6] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reasons. The first 4 lines are from functions areaTailReader / areaNowReader. For these functions a 'scale' argument above 20 is invalid, which is enforced by the fixed length of Info.small arrays. In both functions, as well as in subLeftSumTailReader / subLeftSumNowReader: there is a loop (line 126/141/160/183) where a variable 'depth' iterates from 12 to 'scale', containing an access to array self.small (through line 127/142/161/184), at increasing indices. The index is out of bounds once 'depth' is 19 or 20 (depending on other values), assuming the check on line 121/136/154/177 hasn't already reverted. This reverting also covers branches containing lines 154, 156 in subLeftSumTailReader, and 179, 184, 186 in subLeftSumNowReader. A left shift overflow on any of the 9 lines mentioned is in fact valid, as the bitwise operations remain the intended ones, but will lead to a revert in any case. On lines 151 and 174 (which are identical), the shift overflowing to 0 will lead to reverting due to division by zero, while if the int256 cast overflows the behaviour remains correct as divisibility is not affected by signedness of the divisor. On line 200, the shift overflowing forces the range to be empty (stopIdx <= startIdx), otherwise the revert occurs. For such an empty range, the enclosing branch of function sumRangeTailReader correctly returns zero.

CVF-11 INFO

- **Category** Overflow/Underflow
- **Source** EraBoxcarMidSum.sol

```
30: if (((offset < 0) || (offset >= int256(1 << scale.toUint256())))
    ↪ ) revert OffsetOutOfRange();
```

Description Many instances of the codebase include the left shift and type conversion logic implemented as `int256(1 << scale.toUint256())` or similarly. It performs left shift operations << on integer values and then converts the results to `int256`. This can lead to integer overflows in the following scenarios:

1. If the result of the left shift exceeds the maximum value representable by the integer type, an overflow occurs. This can lead to unexpected and incorrect values.
2. When converting the result of the left shift (which could already be overflowing) to `int256`, further overflow can occur if the value exceeds the maximum limit of `int256`.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[7] for casting operations and using safe operations where possible.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reason. Whether the shift overflows to 0 or type(int256).min, the set of valid 'offset' values becomes empty, making the revert on this line always happen.

[7] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

CVF-12 INFO

- **Category** Overflow/Underflow
- **Source** EraBoxcarMidSum.sol

```
75: int256 length = int256(1 << scale.toUint256());
```

```
98: int256 stopIdx = self.ofBelow ? int256(1 << scale.toUint256()) :
    ↪  (offset + 1);
```

```
114: int256 stopIdx = self.ofBelow ? int256(1 << scale.toUint256())
    ↪ : (offset + 1);
```

Description Many instances of the codebase include the left shift and type conversion logic implemented as `int256(1 << scale.toUint256())` or similarly. It performs left shift operations << on integer values and then converts the results to `int256`. This can lead to integer overflows in the following scenarios:

1. If the result of the left shift exceeds the maximum value representable by the integer type, an overflow occurs. This can lead to unexpected and incorrect values.
2. When converting the result of the left shift (which could already be overflowing) to `int256`, further overflow can occur if the value exceeds the maximum limit of `int256`.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin `SafeCast` library[8] for casting operations and using safe operations where possible.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reasons. In function addRange, when the shift on line 75 overflows, the revert on line 77 can only be avoided when both startIdx and stopIdx are 0, and in this case the function is correctly a no-op. When the shift on line 98/114 overflows in function create / extend, the call to addRange will revert unless 'offset' is -1, in which case the call to coefAddExtend reverts, with an array index 'node' of -1.

[8] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

CVF-13 INFO

- **Category** Overflow/Underflow
- **Source** EraFaberTotals.sol

```
110: Quad nextCoef = smallAtDropReader(self, ((1 << depth.toUint256
   ↪ ()).toInt256() + (stopIdx >> (scale - depth).toUint256())),
   ↪ deadEra);
```

```
112: bool rightChild = ((stopIdx & (1 << ((scale - depth) - 1).
   ↪ toUint256()).toInt256()) != 0);
```

Description There is many instances across the codebase performing left shift operations << on integer values without proper checks for potential overflows. If the result of a left shift exceeds the maximum value representable by the integer type, an overflow occurs, leading to unexpected and incorrect values.

Recommendation Consider mitigating the risk of overflows in left shift operations by using safe operations and/or implementing proper overflow checks.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reasons. These lines are within a loop, where the value of 'depth' iterates from 12 to 'scale'. The call to smallAtDropReader will revert due to array out of bounds access once depth is 19 or 20 (depending on other values), much before overflow would occur on line 110. Therefore also, when overflow is possible on line 112, the function will always revert.

CVF-14 INFO

- **Category** Overflow/Underflow
- **Source** PoolHelper.sol

```
28: return LOG1PPC / fromUint256(1 << uint256(subSplits(splits)));
```

```
36: return int256(1 << uint256(splits));
```

```
126: return day << 11;
```

DescriptionThere is many instances across the codebase performing left shift operations << on integer values without proper checks for potential overflows. If the result of a left shift exceeds the maximum value representable by the integer type, an overflow occurs, leading to unexpected and incorrect values.

Recommendation Consider mitigating the risk of overflows in left shift operations by using safe operations and/or implementing proper overflow checks.

Client Comment No repro. of overflow has been provided - indeed none is possible, for the following reasons. The 'splits' function argument used on line 28 is always pool.splits (an argument is required to provide this pool state to the function). As pool.splits <= 19 is enforced at pool creation (line 32 of InfinityPoolFactory.createPool), and pool.splits does not change thereafter, no overflow is possible. The same argument applies to line 36 (as already laid out in CVF-5). An overflow on line 126 would require a day number trillions of years into the future. This will not arise even in principle, as the source of time is the block timestamp (line 275 in InfinityPool.sol), which is a unit256 in units of seconds, while one era is over 42 seconds long. Note as well, libraries are used in our codebase to satisfy the EVM contract size limit, and do not indicate encapsulation.

CVF-15 INFO

- **Category** Overflow/Underflow
- **Source** GrowthSplitFrame.sol

Description The NewLoan library performs a conversion to `int256` without proper checks for overflow in the following instances:

```
123:    int256 temp = ((scale > int256(MIN_SPLITS)) ? int256(node >>
   ↪  (scale - int256(MIN_SPLITS)).toUint256()) : node);
...
138:    int256 temp = ((scale > int256(MIN_SPLITS)) ? int256(node >>
   ↪  (scale - int256(MIN_SPLITS)).toUint256()) : node);Overflow
   ↪ occurs when a value exceeds the maximum limit of the data type
   ↪ , leading to unexpected behavior and incorrect calculations in
   ↪  subsequent operations.
```

Overflow occurs when a value exceeds the maximum limit of the data type, leading to unexpected behavior and incorrect calculations in subsequent operations.

Recommendation To ensure that type casts cannot overflow and lead to undesirable system behavior, consider using the OpenZeppelin`SafeCast`library[9] for casting operations where possible.

Client Comment No repro. of overflow has been provided - indeed none is possible, as neither subtracting a positive value (`MIN_SPLITS` is 12) nor right shifting can overflow. Underflow is also precluded as the toUint256 call happens only in the scale > `MIN_SPLITS` case, whereby scale - `MIN_SPLITS` is positive. This applies to both lines, which are identical.

[9] https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast

CVF-16 INFO

- **Category** Unclear behavior
- **Source** DeadlineHelper.sol

```
5: function jumpBitLength(int256 num) pure returns (int8) {
```

**Description **The `jumpBitLength` function in the `DeadlineHelper.sol` file calculates the bit length of an input integer `num`. However, it lacks input validation to ensure that the argument value falls within the range that the function can correctly handle. The function's logic relies on bitwise operations and comparisons that are only valid for values between 0 and $2^16 - 1 (0 to 65535)$ If the argument exceeds this range, the function's behavior becomes undefined. It might return incorrect results, revert due to unexpected conditions, or even lead to unintended side effects in the contract's state.

Recommendation Add input validation at the beginning of the `jumpBitLength` function to ensure that the input value `num` is within the valid range. This can be done using a `require` statement:

```
require(num >= 0 && num <= 65535, "Input␣out␣of␣range");
```

Client Comment This code works correctly and as intended. The ouput of this function is capped at 13. As such inputs either above 2^12, or negative, must return 13, and indeed correctly do so.

CVF-25 FIXED

- **Category** Suboptimal
- **Source** Constants.sol

```
26: ///@dev TODO: see if storing bytes array is effiecient or
   ↪ storing all of them separately is effiecient
```

Description The `DEFLATOR_STEPS` constant array is defined as a `bytes` array, which is stored directly in the bytecode of the contract. However, when accessing individual elements of this array, the entire array is copied into memory, leading to inefficient memory usage. The cost of extracting a single value from the array is proportional to the size of the array, making it gas-expensive.

Recommendation Consider packing two `bytes16` values into a single `bytes32` word. This will reduce the size of the array by half. Then, implement a binary search tree structure to store the packed values. This will allow for efficient retrieval of individual values based on their index.

Client Comment Optimised as suggested.

CVF-26 INFO

- **Category** Suboptimal
- **Source** JumpyFallback.sol

```
39: int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

```
78: int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

```
117: int256 cursor = ((1 << splits.toUint256()).toInt256() + offset)
   ↪ ;
```

Recommendation Changing the type for this variable to "uint256" would made code simpler and more efficient.

CVF-27 INFO

- **Category** Suboptimal
- **Source** DropFaberTotals.sol

```
30: if (!((scale.toUint256() >= MIN_SPLITS && ((start >= 0)) && ((
   ↪ start.toUint256() + areas.length) <= (1 << scale.toUint256()))
   ↪ ))) {
```

```
34: Quad[] memory carry = new Quad[](scale.toUint256());
```

```
36: int256 first = ((1 << scale.toUint256()) + start.toUint256()).
   ↪ toInt256();
```

```
44: for (int256 index = 0; (index.toUint256() < areas.length); index
   ↪  = (index + 1)) {
```

```
46: change = areas[index.toUint256()];
```

```
48: while ((depth.toUint256() >= MIN_SPLITS)) {
```

```
53:           carry[depth.toUint256()] = change;
```

```
56:          smallAdd(self, poolEra, node, (carry[depth.toUint256()]
   ↪ - change), aDeadEra);
57:           change = (change + carry[depth.toUint256()]);
```

Description The code within the `DropFaberTotals`, `EraFaberTotals`, and `GrowthSplitFrame` libraries frequently converts between signed `int256` and unsigned `uint256` integer types. While these conversions might seem harmless, they can

introduce subtle errors, decrease code readability, and potentially increase gas fees due to unnecessary type conversions.

Recommendation Consider refactoring the code to minimize unnecessary type conversions. Here are some specific suggestions:

1. Choose appropriate types: If a variable is never expected to hold negative values, declare it as `uint256` from the beginning to avoid type conversions. For example, `scale` and `depth` variables might be declared as `uint256` as they seem to never hold negative values.
2. Use unsigned operations: When performing arithmetic or bitwise operations on variables that are inherently non-negative, use unsigned versions of the operators. For instance, replace `1 << scale.toUint256()` with `1 << scale` if `scale` is of type `uint256`.

CVF-28 INFO

- **Category** Suboptimal
- **Source** EraFaberTotals.sol

Description See CVF-27.

```
39:     t2 = (1 << pool.splits.toUint256()).toInt256();
```

```
46: DeadlineSet.Info storage temp = self.small[node.toUint256()];
```

```
51: if (!(((scale <= MIN_SPLITS.toInt256() && ((0 <= startIdx)) && (
   ↪ startIdx <= stopIdx)) && (stopIdx <= (1 << scale.toUint256()).
   ↪ toInt256())))) {
```

```
57: int256 start = ((1 << scale.toUint256()).toInt256() + startIdx);
```

```
59: int256 stop = ((1 << scale.toUint256()).toInt256() + stopIdx);
```

```
80: if ((scale <= MIN_SPLITS.toInt256())) {
```

```
83:     if (!((((0 <= startIdx) && (startIdx <= stopIdx)) && (stopIdx
   ↪  <= (1 << scale.toUint256()).toInt256())))) {
```

```
89:         MIN_SPLITS.toInt256(),
        (startIdx >> (scale - MIN_SPLITS.toInt256()).toUint256()),
        (stopIdx >> (scale - MIN_SPLITS.toInt256()).toUint256()),
```

```
100: if (((stopIdx % (1 << (scale - MIN_SPLITS.toInt256()).toUint256
   ↪ ()).toInt256()) == 0)) {
```

CVF-29 INFO

- **Category** Suboptimal
- **Source** GrowthSplitFrame.sol

```
36: self.activeLeaf = int256(1 << pool.splits.toUint256()) + initBin
   ↪ ;
```

```
65: if (!(((scale <= int256(MIN_SPLITS) && (0 <= startIdx)) && (
   ↪ stopIdx <= int256(1 << scale.toUint256())))) revert
   ↪ InvalidSumTotalArguments();
```

```
68: int256 start = (int256(1 << scale.toUint256()) + startIdx);
int256 stop = (int256(1 << scale.toUint256()) + stopIdx);
```

```
96: return sumTotal(self, poolEra, poolDeflator, int256(MIN_SPLITS),
   ↪  startTub, stopTub, largeAtAccruedReader);
```

```
101: int256 activeTub = (int256(self.activeLeaf - int256(1 << pool.
   ↪ splits.toUint256())) >> (pool.splits - int256(MIN_SPLITS)).
   ↪ toUint256());
```

```
121: if (!((offset & int256(-1 << scale.toUint256())) == 0))
   ↪ revert InvalidAreaArguments();
122: int256 node = (int256(1 << scale.toUint256()) + offset);
123: int256 temp = ((scale > int256(MIN_SPLITS)) ? int256(node >> (
   ↪ scale - int256(MIN_SPLITS)).toUint256()) : node);
```

Description See CVF-27.

CVF-30 FIXED

- **Category** Suboptimal
- **Source** NewLoan.sol

```
176: assert(!params.tokenMix.isNaN() || !params.lockinEnd.isNaN());
```

Description The newLoan function in the NewLoan library uses an `assert` statement to check if either `tokenMix` or `lockinEnd` is not NaN (Not a Number). While this check is important to ensure valid input, using `assert` is not appropriate in this context.

In Solidity, `assert` is intended for internal checks that should never fail. If an `assert` statement fails, it indicates a critical error in the contract's logic. In this case, if either `tokenMix` or `lockinEnd` is NaN, it's likely due to invalid input rather than a bug in the contract itself.

Recommendation Replace the `assert` statement with a `require` statement. `require` is designed for input validation and conditional checks. If a `require` statement fails, it reverts the transaction with an error message, which is the desired behavior for invalid input.

Client Comment Changed to use a custom error.

CVF-31 INFO

- **Category** Suboptimal
- **Source** Advance.sol

```
154: vars.upward = (vars.surplus0 < vars.surplus1);
```

```
161: if (vars.upward) {
    vars.freed1 = (vars.freed1 - ((pool.epsilon * vars.needed) *
       ↪ sqrtStrike(pool.splits, pool.tickBin)));
```

```
164:    vars.freed0 = (vars.freed0 + (((vars.qMove * pool.epsilon) *
   ↪  vars.liquidity) / sqrtStrike(pool.splits, pool.tickBin)));
165:    vars.freed1 = (vars.freed1 - (((vars.qMove * pool.epsilon) *
   ↪  vars.liquidity) * sqrtStrike(pool.splits, pool.tickBin)));
```

```
167:   vars.freed0 = (vars.freed0 - ((pool.epsilon * vars.needed) /
   ↪ sqrtStrike(pool.splits, pool.tickBin)));
```

```
169   vars.freed1 = (vars.freed1 + (((vars.qMove * pool.epsilon) *
   ↪ vars.liquidity) * sqrtStrike(pool.splits, pool.tickBin)));
170:    vars.freed0 = (vars.freed0 - (((vars.qMove * pool.epsilon) *
   ↪  vars.liquidity) / sqrtStrike(pool.splits, pool.tickBin)));
```

```
172: while ((((vars.upward) ? vars.freed0 : vars.freed1) <
   ↪ POSITIVE_ZERO)) {
```

```
181:    if (vars.upward) {
        vars.freed0 =
```

```
184:        vars.freed1 =
```

```
187:        vars.freed1 =
```

```
189:        vars.freed0 =
```

```
197:        POSITIVE_ONE - (((vars.freed0 / pool.epsilon) / vars.
   ↪ liquidity) * sqrtStrike(pool.splits, pool.tickBin)),
```

```
201:    vars.freed1 = vars.freed1
        + (((POSITIVE_ONE - pool.binFrac) * pool.epsilon) * vars.
           ↪ liquidity) * sqrtStrike(pool.splits, pool.tickBin);
202:    pool.surplus1 = pool.surplus1 + vars.freed1;
```

```
206:        min(((vars.freed1 / pool.epsilon) / vars.liquidity) /
   ↪ sqrtStrike(pool.splits, pool.tickBin), POSITIVE_ONE);
```

```
208:    vars.freed0 = vars.freed0 + ((pool.binFrac * pool.epsilon) *
   ↪  vars.liquidity) / sqrtStrike(pool.splits, pool.tickBin);
209:    pool.surplus0 = pool.surplus0 + vars.freed0;
```

Description The `advance` function uses the `vars.upward` flag to conditionally swap the values of `vars.freed0` with `vars.freed1` and `vars.surplus0` with `vars.surplus1`. This is done using multiple conditional statements (`if/else`) and ternary operators, which can make the code harder to read and potentially less gas efficient.

Recommendation Consider refactoring the code to directly swap the semantics of `freed0` with `freed1` and `surplus0` with `surplus1` based on the value of the `vars.upward` flag. This approach eliminates the need for conditional branching and makes the code more concise and potentially more efficient.

CVF-32 INFO

- **Category** Suboptimal
- **Source** Swapper.sol

```
140: amounts[0] = pool.epsilon * liquidityRatio / sqrtStrike(pool.
   ↪ splits, bin);
```

```
143: amounts[0] = pool.epsilon * liquidityRatio * sqrtStrike(pool.
   ↪ splits, bin);
```

Description The following `sqrtStrike` expressions are calculated more than once within the `Swapper` library, even though its values remain unchanged. This happens within the instances of:

```
sqrtStrike(pool.splits, self.startBin + int256(i)
sqrtStrike(pool.splits, bin)
sqrtStrike(pool.splits, self.strikeBin)
sqrtStrike(pool.splits, self.startBin + int256(index))
```

This redundancy leads to an increase of gas fees, especially when it happens within the loops.

Recommendation Consider calculating the value of `sqrtStrike` once and storing it in a temporary variable where possible.

CVF-33 INFO

- **Category** Suboptimal
- **Source** Advance.sol

```
127:    (vars.freed0 - ((((POSITIVE_ONE - pool.binFrac) * pool.
   ↪ epsilon) * vars.needed) / sqrtStrike(pool.splits, pool.tickBin
   ↪ )));
```

```
129: vars.surplus1 = (vars.freed1 - (((pool.binFrac * pool.epsilon)
   ↪ * vars.needed) * sqrtStrike(pool.splits, pool.tickBin)));
```

```
164:         vars.freed0 = (vars.freed0 + (((vars.qMove * pool.
   ↪ epsilon) * vars.liquidity) / sqrtStrike(pool.splits, pool.
   ↪ tickBin)));
165:         vars.freed1 = (vars.freed1 - (((vars.qMove * pool.
   ↪ epsilon) * vars.liquidity) * sqrtStrike(pool.splits, pool.
   ↪ tickBin)));
```

```
169:         vars.freed1 = (vars.freed1 + (((vars.qMove * pool.
   ↪ epsilon) * vars.liquidity) * sqrtStrike(pool.splits, pool.
   ↪ tickBin)));
170:         vars.freed0 = (vars.freed0 - (((vars.qMove * pool.
   ↪ epsilon) * vars.liquidity) / sqrtStrike(pool.splits, pool.
   ↪ tickBin)));
```

```
183:                 vars.freed0 + ((pool.epsilon * (vars.liquidity
   ↪ + vars.needed)) / sqrtStrike(pool.splits, pool.tickBin));
```

```
185:                 vars.freed1 - ((pool.epsilon * (vars.liquidity
   ↪ + vars.needed)) * sqrtStrike(pool.splits, pool.tickBin));
```

```
188:                 vars.freed1 + ((pool.epsilon * (vars.liquidity
   ↪ + vars.needed)) * sqrtStrike(pool.splits, pool.tickBin));
```

```
190:                 vars.freed0 - ((pool.epsilon * (vars.liquidity
   ↪ + vars.needed)) / sqrtStrike(pool.splits, pool.tickBin));
```

```
197:             POSITIVE_ONE - (((vars.freed0 / pool.epsilon) /
   ↪ vars.liquidity) * sqrtStrike(pool.splits, pool.tickBin)),
```

```
202:             + (((POSITIVE_ONE - pool.binFrac) * pool.epsilon) *
   ↪  vars.liquidity) * sqrtStrike(pool.splits, pool.tickBin);
```

```
208:         vars.freed0 = vars.freed0 + ((pool.binFrac * pool.
   ↪ epsilon) * vars.liquidity) / sqrtStrike(pool.splits, pool.
   ↪ tickBin);
```

Description The expression `sqrtStrike(pool.splits, pool.tickBin)` is repeatedly calculated within the `for` loop and the nested `while` loop in the `advance` function within the `Advance` library. This calculation is performed multiple times, even though the values of `pool.splits` and `pool.tickBin` remain unchanged throughout the loops. This redundancy can lead to an increase in gas fees, especially if the loops iterate many times.

Recommendation To optimize the code, calculate the value of `sqrtStrike(pool.splits, pool.tickBin)` once before each loop and store it in a temporary variable:

```
Quad sqrtMid = sqrtStrike(pool.splits, pool.tickBin); // Calculate
   ↪ once
```

Then, use this variable within the loops, instead of recalculating the expression.

CVF-34 INFO

- **Category** Suboptimal
- **Source** Swapper.sol

```
111: sum1 = sum1 + self.owed[i] / sqrtStrike(pool.splits, self.
   ↪ startBin + int256(i));
112: sum2 = sum2 + self.owed[i] * sqrtStrike(pool.splits, self.
   ↪ startBin + int256(i));
```

```
121: sum1 = sum1 + self.lent[i] / sqrtStrike(pool.splits, self.
   ↪ startBin + int256(i));
122: sum2 = sum2 + self.lent[i] * sqrtStrike(pool.splits, self.
   ↪ startBin + int256(i));
```

Description See CVF-32.

CVF-35 INFO

- **Category** Suboptimal
- **Source** Swapper.sol

```
300: amounts[index] = signed(enable, -self.owed[index] / sqrtStrike(
   ↪ pool.splits, self.startBin + int256(index)));
```

```
304: amounts[index] = signed(enable, -self.owed[index] * sqrtStrike(
   ↪ pool.splits, self.startBin + int256(index)));
```

Description See CVF-32.

CVF-36 INFO

- **Category** Suboptimal
- **Source** Swapper.sol

```
196:                ? pool.epsilon / sqrtStrike(pool.splits, self.
    ↪ strikeBin)
197:             : pool.epsilon * sqrtStrike(pool.splits, self.
    ↪ strikeBin)
```

```
202:     * (fixedToken == Z ? pool.epsilon / sqrtStrike(pool.splits,
    ↪ self.strikeBin) : pool.epsilon * sqrtStrike(pool.splits, self.
    ↪ strikeBin));
```

```
250:         vars.capacity0 = (vars.owedMean / sqrtStrike(pool.
    ↪ splits, self.strikeBin)) + (vars.deadDeflator * self.
    ↪ lentCapacity0);
251:        vars.capacity1 = (vars.owedMean * sqrtStrike(pool.splits
    ↪ , self.strikeBin)) + (vars.deadDeflator * self.lentCapacity1);
```

```
261:         (token == Z) ? (POSITIVE_ONE / sqrtStrike(pool.splits,
    ↪ self.strikeBin)) : sqrtStrike(pool.splits, self.strikeBin);
```

```
404: * ((token == Z) ? (pool.epsilon / sqrtStrike(pool.splits, self.
    ↪ strikeBin)) : (pool.epsilon * sqrtStrike(pool.splits, self.
    ↪ strikeBin)))
```

```
459:     ? (vars.owedMean / sqrtStrike(pool.splits, self.strikeBin))
460:        : (vars.owedMean * sqrtStrike(pool.splits, self.
    ↪ strikeBin));
```

```
465: ? (pool.epsilon / sqrtStrike(pool.splits, self.strikeBin))
466: : (pool.epsilon * sqrtStrike(pool.splits, self.strikeBin));
```

Description See CVF-32.

CVF-37 FIXED

- **Category** Unclear behavior
- **Source** BucketRolling.sol

```
18: self.ring[i] = initial / POSITIVE_EIGHT;
```

```
27: self.ring[i] = initial / POSITIVE_EIGHT;
```

Description In the `init` functions (both of them) within the `BucketRolling` library, the expression `initial / POSITIVE_EIGHT` is redundantly calculated within the `for` loop. This calculation is performed on every iteration, even though the result remains the same throughout the loop. This unnecessary repetition can lead to a slight increase in gas, especially if the loop iterates many times.

Recommendation To optimize the code, calculate the value of `initial / POSITIVE_EIGHT` once before the loop and store it in a temporary variable. Then, use this variable within the loop instead of recalculating the expression.

Client Comment Optimised as suggested.

CVF-38 INFO

- **Category** Unclear behavior
- **Source** NewLoan.sol

```
55: return sqrtStrike(splits, bin) - strike / sqrtStrike(splits, bin
   ↪ );
```

Description The expression `sqrtStrike(splits, bin)` is calculated twice within the `tilter` functions, even though the values of `splits` and `bin` remain unchanged throughout the loops.

Recommendation To optimize the code, calculate the value of `sqrtStrike(splits, bin)` once and store it in a temporary variable.

CVF-39 INFO

- **Category** Unclear behavior
- **Source** SparseFloat.sol

```
59: temp := shr(15, val)
```

Description The value calculated here is basically a constant.

Recommendation Consider just hardcoding it, rather than calculating.

CVF-40 INFO

- **Category** Unclear behavior
- **Source** LP.sol

```
71: function init(int256 startTub, int256 stopTub, Quad liquidity,
    ↪ Stage stage, int256 earnEra) internal view returns (Info
    ↪ memory) {
```

Description The `init` function in the LP library initializes a new liquidity position (LP) with parameters such as `startTub`, `stopTub`, and `liquidity`. However, it lacks input validation to ensure that these arguments are valid and consistent with the expected behavior of the contract. Specifically, there are no checks to ensure that:

- `0 <= startTub <= stopTub <= TUBS`: This condition ensures that the start and stop tubs are within the valid range of tubs in the pool.
- `liquidity > 0`: This condition ensures that the liquidity being added is positive.

If these conditions are not met, it could lead to unexpected behavior, incorrect calculations, and potential vulnerabilities in the contract.

Recommendation Consider adding input validations at the beginning of the `init` function. For instance, add:

```
require(0 <= startTub && startTub <= stopTub && stopTub <= TUBS, "
    ↪ Invalid␣tub␣range");
require(liquidity > 0, "Liquidity␣must␣be␣above␣0");
```

This will ensure that the LP is initialized with valid parameters, preventing potential errors and vulnerabilities.

CVF-41 INFO

- **Category** Unclear behavior
- **Source** GrowthSplitFrame.sol

```
120: function areaTailReader(Info storage self, int256 scale, int256
    ↪  offset) internal view returns (Quad) {
```

```
135: function areaNowReader(Info storage self, PoolState storage
    ↪ pool, int256 scale, int256 offset) internal returns (Quad) {
```

```
150: function subLeftSumTailReader(Info storage self, int256 scale,
    ↪ int256 stopIdx) internal view returns (Quad) {
```

```
173: function subLeftSumNowReader(Info storage self, PoolState
   ↪ storage pool, int256 scale, int256 stopIdx) internal returns (
   ↪ Quad) {
```

```
196: function sumRangeTailReader(Info storage self, PoolState
   ↪ storage pool, int256 scale, int256 startIdx, int256 stopIdx)
   ↪ internal returns (Quad) {
```

Description Multiple functions within different libraries lack input validation for the `scale` argument. This argument is usually used to determine the size of data structures and perform bitwise operations. If the `scale` value is too small or too large (less than `MIN_SPLITS` or more than `MAX_SPLITS`), it can lead to out-of-bounds errors, incorrect calculations, and potentially unexpected behavior in the contract.

Recommendation Consider adding input validation at the beginning of each function to ensure that the `scale` value is within the allowed range.

CVF-42 INFO

- **Category** Unclear behavior
- **Source** EraBoxcarMidSum.sol

```
29: function apply_(Info storage self, int256 poolEra, int256 scale,
   ↪  int256 offset) internal returns (Quad) {
```

Description See CVF-41.

CVF-43 INFO

- **Category** Unclear behavior
- **Source** EraBoxcarMidSum.sol

```
67:     int256 scale,
```

```
96: function create(Info storage self, PoolState storage pool, Quad
   ↪ amount, int256 scale, int256 offset, OptInt256 deadEra)
   ↪ internal {
```

Description See CVF-41.

CVF-44 INFO

- **Category** Unclear behavior
- **Source** JumpyAnchorFaber.sol

```
270: function addCreateWriter(Info storage self, int256 poolEra,
   ↪ int256 scale, int256 start, Quad[] memory areas, OptInt256
   ↪ deadEra) internal {
```

```
357: function addCreateWriter(Info storage self, int256 poolEra,
   ↪ int256 scale, int256 offset, Quad area, OptInt256 deadEra)
   ↪ internal {
```

```
389: function subLeftSumNowReader(Info storage self, int256 poolEra,
   ↪  int256 scale, int256 stopIdx) internal returns (Quad) {
```

```
422: function sumRangeNowReader(Info storage self, int256 poolEra,
   ↪ int256 scale, int256 startIdx, int256 stopIdx) internal
   ↪ returns (Quad) {
```

```
442: function addExpireWriter(Info storage self, int256 poolEra,
   ↪ int256 scale, int256 offset, Quad area, int256 deadEra)
   ↪ internal {
```

```
473: function create(Info storage self, PoolState storage pool, Quad
   ↪ [] memory amounts, int256 scale, int256 start, OptInt256
   ↪ deadEra) internal {
```

```
523: function sumTotalNowReader(Info storage self, int256 poolEra,
   ↪ int256 scale, int256 startIdx, int256 stopIdx) internal
   ↪ returns (Quad) {
```

Description See CVF-41.

CVF-45 INFO

- **Category** Unclear behavior
- **Source** GrowthSplitFrame.sol

```
40: function live(Info storage self, int256 splits) internal view
   ↪ returns (PiecewiseGrowthNew.Info storage) {
```

Description Multiple functions within the `JumpyFallback` and `GrowthSplitFrame` libraries lack input validation for the `splits` argument. This argument is used to calculate array indices and perform bitwise operations. If the `splits` value is too small (less than `MIN_SPLITS`) or too large (greater than `MAX_SPLITS`), it can lead to out-of-bounds errors, incorrect calculations, and potentially unexpected behavior in the contract.

Recommendation Consider adding input validation at the beginning of each applicable function to ensure that the `splits` value is within the allowed range. This can be done using a `require` statement:

```
require(splits >= MIN_SPLITS && splits <= MAX_SPLITS, "Splits␣out␣of
   ↪ ␣range");
```

CVF-46 INFO

- **Category** Unclear behavior
- **Source** JumpyFallback.sol

```
36: function nowAt(Info storage self, int256 poolEra, int256 splits,
   ↪  int256 offset) internal returns (Quad) {
```

```
75: function dropAt(Info storage self, int256 splits, int256 offset,
   ↪  int256 deadEra) internal returns (Quad) {
```

```
114: function enter(Info storage self, int256 poolEra, int256 splits
   ↪ , int256 offset) internal returns (DeadlineShaded.Info storage
   ↪ ) {
```

Description Multiple functions within the `JumpyFallback` and `GrowthSplitFrame` libraries lack input validation for the `splits` argument. This argument is used to calculate array indices and perform bitwise operations. If the `splits` value is too small (less than `MIN_SPLITS`) or too large (greater than `MAX_SPLITS`), it can lead to out-of-bounds errors, incorrect calculations, and potentially unexpected behavior in the contract.

Recommendation Consider adding input validation at the beginning of each applicable function to ensure that the `splits` value is within the allowed range. This can be done using a `require` statement:

```
require(splits >= MIN_SPLITS && splits <= MAX_SPLITS, "Splits␣out␣of
   ↪ ␣range");
```

CVF-47 INFO

- **Category** Unclear behavior
- **Source** EraFaberTotals.sol

```
103: if (!(scale >= MIN_SPLITS.toInt256() && (((stopIdx & (-1 <<
   ↪ scale.toUint256())) == 0)))) revert InvalidSubLeftSumArguments
   ↪ ();
```

Description See CVF-41.

CVF-48 INFO

- **Category** Unclear behavior
- **Source** BytesLib.sol

```
14: require(_length + 31 >= _length, "slice_overflow");
15: require(_start + _length >= _start, "slice_overflow");
```

Description The `slice` function in the `BytesLib` library includes two overflow checks using `require` statements:

```
14:    require(_length + 31 >= _length, "slice_overflow");
15:    require(_start + _length >= _start, "slice_overflow");
```

These checks are intended to prevent overflows when calculating the end position of the slice. However, they are redundant because all Solidity compiler versions above 0.8.0 already perform overflow checks by default. If an overflow were to occur in these additions, the compiler would automatically revert the transaction, making the `require` statements unnecessary.

Recommendation Consider placing the abovementioned `require` checks within an `unchecked` block to bypass the compiler's default overflow checks:

```
14:     unchecked {
15:         require(_length + 31 >= _length, "slice_overflow"); //
   ↪ Redundant check
16:         require(_start + _length >= _start, "slice_overflow"); //
   ↪  Redundant check
17:     }
```

CVF-49 FIXED

- **Category** Procedural
- **Source** BoxcarTubFrame.sol

```
50: function addRange(Info storage self, int256 startIdx, int256
   ↪ stopIdx, Quad change) internal {
```

```
78: function add(Info storage self, int256 startTub, int256 stopTub,
   ↪  Quad change) internal {
```

Description These functions are identical.

Recommendation Consider removing one of them.

Client Comment Removed one as suggested.

CVF-50 FIXED

- **Category** Procedural
- **Source** BoxcarTubFrame.sol

```
54: function addRange(Info storage self, int256 scale, int256
   ↪ startIdx, int256 stopIdx, Quad change) internal {
```

```
74: function add(Info storage self, int256 scale, int256 startIdx,
   ↪ int256 stopIdx, Quad change) internal {
```

Description These functions are identical.

Recommendation Consider removing one of them.

Client Comment Removed one as suggested.

CVF-51 INFO

- **Category** Procedural
- **Source** OptInt256.sol

```
14: function isDefined(OptInt256 x) pure returns (bool) {
```

```
18: function neq(OptInt256 x, OptInt256 y) pure returns (bool) {
```

```
22: function unwrap(OptInt256 x) pure returns (int256 result) {
```

```
27: function wrap(int256 x) pure returns (OptInt256 result) {
```

Description These top-level functions have names that are too generic and may easily clash with other names.

Recommendation Consider using more specific names or moving these functions into a library.

CVF-52 INFO

- **Category** Suboptimal
- **Source** BoxcarTubFrame.sol

```
23: function coefAt(Info storage self, int256 node) internal view
   ↪ returns (Quad) {
```

```
27: function coefAdd(Info storage self, int256 node, Quad amount)
   ↪ internal {
```

```
31: function apply_(Info storage self, int256 scale, int256 offset)
   ↪ internal view returns (Quad) {
```

```
42: function apply_(Info storage self, int256 offset) internal view
   ↪ returns (Quad) {
```

Description The functions `coefAt`, `coefAdd`, `apply_`, and `apply_` in the `BoxcarTubFrame` library are declared as `internal`, which means they can be called by any contract that inherits this library or within the library itself. However, these functions are not intended to be part of the library's public interface and are only used internally within the library.

Recommendation Consider declaring the abovementioned functions as `private` instead of `internal`. This will restrict their access to only within the `BoxcarTubFrame` library, preventing them from being called externally. This is a best practice for functions that are not meant to be public and reduces the potential attack surface.

CVF-53 INFO

- **Category** Suboptimal
- **Source** Utils.sol

```
5:  if (shift < 128) return (value >> shift) & int128(int256((1 <<
   ↪ (128 - shift)) - 1));
return 0;
```

Description The `int128ShiftRightUnsigned` function in the `Utils.sol` file aims to perform an unsigned right shift on `int128`. However, the current implementation is more complex than necessary:

```
4:      function int128ShiftRightUnsigned(int128 value, uint256
   ↪ shift) pure returns (int128) {
5:          if (shift < 128) return (value >> shift) & int128(int256
   ↪ ((1 << (128 - shift)) - 1));
6:          return 0;
```

It involves bitwise operations and type conversions that can be simplified.

Recommendation Consider replacing the current implementation of `int128ShiftRightUnsigned` with the following simplified version:

```
function int128ShiftRightUnsigned(int128 value, uint256 shift) pure
   ↪ returns (int128) {
    return int128(uint128(value) >> shift);
}
```

This eliminates the need for the conditional check and the masking operation, resulting in cleaner and more efficient code.

CVF-54 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

```
24: Quad latest;
```

Recommendation This field is not used in this library and should be moved to the libraries where it is actually being used.

CVF-55 INFO

- **Category** Unclear behavior
- **Source** SparseFloat.sol

```
44: function unpack(Quad quad) internal pure returns (FloatBits.Info
   ↪  memory) {
```

Description The `unpack` function in the `QuadPacker` library is designed to convert a `Quad` value (a fixed-point number representation) into a `FloatBits.Info` struct, which represents the floating-point components (exponent and significand). However, the current implementation does not correctly handle negative `Quad` values. It attempts to convert the `Quad` value directly to an unsigned integer using `uint128(quad.unwrap())`, which will produce incorrect results for negative inputs.

Recommendation Modify the `unpack` function to correctly handle negative `Quad` values.

CVF-56 INFO

- **Category** Suboptimal
- **Source** SparseFloat.sol

```
139:     carry = oldLow < 0 == lowPart < 0 ? oldLow < 0 : !(newLow <
   ↪ 0);
```

```
149: carry = oldHigh < 0 == highPart < 0 ? oldHigh < 0 : !(newHigh <
   ↪  0);
```

Description The `add` function in the `SparseFloat` library uses a complex calculation to determine if a carry is needed when adding two numbers. The current implementation involves comparing the signs of the input values and the result, which is unnecessary and can be simplified.

Recommendation Consider directly calculating the new value as a `uint256`, store the lower 128 bits as the result, and use the 129th bit (the most significant bit of the upper 128 bits) as the carry. By simplifying the carry calculation, you can improve the code's readability and potentially its performance by avoiding unnecessary comparisons and branching.

CVF-57 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

```
319: while ((pastJump < self.deadlineJumps.nextJump)) {
320:    int256 eraStep = 0;

321:    if ((pastJump == 0)) eraStep = 0;
322:  else if (((pastEra & (1 << (pastJump.toUint256() - 1)).
   ↪ toInt256()) != 0)) eraStep = (pastJump - 1);
323:    else eraStep = pastJump;

324:    pastEra = (pastEra + (1 << eraStep.toUint256()).toInt256());
325:    nextDeflator = (nextDeflator * wrap(bytes16(DEFLATOR_STEPS.
   ↪ slice((eraStep * 32).toUint256(), 16))));
326:    pastJump = (pastJump + 1);
}
```

Description The logic within the `while` loop in the `growAccruing` function of the `PiecewiseGrowthNew` library is unnecessarily complex. It attempts to calculate the next deflator value by iteratively applying deflation steps based on the `pastJump` and `pastEra` variables. This approach is inefficient and can be simplified.

By replacing the `while` loop with a direct calculation, you can significantly improve the code's readability and efficiency.

Recommendation Consider directly computing the next deflator value based on the `nextJump` value. This can be achieved by implementing an optimized function (e.g., `calculateDeflatorForEra`) that takes the era as input and returns the corresponding deflator value.

CVF-58 INFO

- **Category** Suboptimal
- **Source** BytesLib.sol

```
35: let lengthmod := and(_length, 31)
```

Description The `slice` function in the `BytesLib` library uses a variable named `lengthmod` to handle the case when the slice length is not a multiple of 32 bytes. This variable is calculated as `and(_length, 31)`, which gives the remainder of the slice length when divided by 32. The code then uses this variable to adjust the starting position of the copy operation and to handle partial word reads. However, this logic is unnecessarily complex and can be simplified.

Recommendation Instead of using `lengthmod`, consider directly start copying from the beginning of the slice and copy as many 32-byte words as needed to cover the entire slice length. This approach might copy some extra bytes after the slice, but this is acceptable because the memory allocated for the slice array is padded to 32 bytes, and unallocated memory can contain arbitrary data.

CVF-59 INFO

- **Category** Procedural
- **Source** Constants.sol

```
61: Quad constant LOG1PPC = Quad.wrap(0
   ↪ x3ff8460d6ccca3676b71e159f1d244a4);
```

Description The value of this constant is ln(1.01), while according to documentation provided along with the audit, it should be ln(1.0001).

CVF-60 FIXED

- **Category** Procedural
- **Source** IInfinityPoolState.sol

```
36: int256 tickSpacing;
```

Description This field isn't used.

Recommendation Consider removing it.

Client Comment Done, as suggested.

8 Minor Issues

CVF-61 FIXED

- **Category** Procedural
- **Source** InfinityPool.sol

Recommendation Typo, “prforms”.

Client Comment Corrected typo.

```
138 * @dev This function allows the trader to modify the desired
    ↪ exposure to tokens and prforms quite significant changes into
    ↪ the internal accounting of the pool.
```

CVF-62 INFO

- **Category** Procedural
- **Source** IInfinityPoolState.sol

Description In ERC20, the “decimals” property is used by UI to render token amounts in human readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

```
34 Quad tenToPowerDecimals0;
   Quad tenToPowerDecimals1;
```

CVF-63 INFO

- **Category** Documentation
- **Source** EachPayoff.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or documenting.

```
25 function capperBegin(PoolState storage pool, int256 pivotTick, bool
    ↪ side, int128 exponent) internal returns (int128) {
```

CVF-64 FIXED

- **Category** Suboptimal
- **Source** InfinityPool.sol

Description The value “logBin(pool.splits)” is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment Optimised as suggested.

```
45 pool.epsilon = expm1(logBin(pool.splits) / POSITIVE_FOUR) - expm1(-
      ↪ logBin(pool.splits) / POSITIVE_FOUR);
   pool.fee = -expm1(-logBin(pool.splits) / POSITIVE_TWO);
```

```
49     / ((logBin(pool.splits) / POSITIVE_FOUR).exp() + (-logBin(pool.
          ↪ splits) / POSITIVE_FOUR).exp());
```

CVF-65 FIXED

- **Category** Procedural
- **Source** GapStagedFrame.sol

Description These variables have the same value.

Recommendation Consider removing once of them.

Client Comment Done, as suggested.

```
90 Gap.Info storage idx2 = self.gaps[idx1.toUint256()];
```

```
93 Gap.Info storage temp = self.gaps[idx1.toUint256()];
```

CVF-66 FIXED

- **Category** Procedural
- **Source** GapStagedFrame.sol

Description These variables have the same value.

Recommendation Consider removing once of them.

Client Comment Done, as suggested.

```
98  Gap.Info storage idx0 = self.gaps[0];
```

```
104 Gap.Info storage temp0 = self.gaps[0];
```

CVF-67 FIXED

- **Category** Procedural
- **Source** GapStagedFrame.sol

Description These variables have the same value.

Recommendation Consider removing once of them.

Client Comment Done, as suggested.

```
100 Gap.Info storage idx1 = self.gaps[1];
```

```
106 Gap.Info storage temp1 = self.gaps[1];
```

CVF-68 INFO

- **Category** Readability
- **Source** GapStagedFrame.sol

Description Using several local variables with the same name inside one function makes code harder to read.

Recommendation Consider using different names.

```
82  int256 temp = (eraDay(pool.era) - self.gapDay);
```

```
93          Gap.Info storage temp = self.gaps[idx1.toUint256()];
```

CVF-69 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Brackets are redundant in the condition.

```
45 if (((bucket - self.lastBucket) >= 9)) {
```

CVF-70 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Brackets are redundant.

```
44 cursor = (cursor >> 1);
```

```
57 temp = !(self.tree[cursor.toUint256()].deadlineJumps.touched);
```

```
67 leaf.deadlineJumps.latest = (leaf.deadlineJumps.latest + trunk.
   ↪ deadlineJumps.latest);
```

```
83 cursor = (cursor >> 1);
```

```
96 temp = !(self.tree[cursor.toUint256()].deadlineJumps.touched);
```

```
106 leaf.deadlineJumps.latest = (leaf.deadlineJumps.latest + trunk.
    ↪ deadlineJumps.latest);
```

```
125 cursor = (cursor >> 1);
```

```
131 temp = !(self.tree[cursor.toUint256()].deadlineJumps.touched);
```

```
141 cursor = (cursor >> 1);
```

CVF-71 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Brackets are redundant.

```
129     subArea = (rightChild ? (subArea - nextCoef) : (subArea +
           ↪ nextCoef));
```

```
144     subArea = (rightChild ? (subArea - nextCoef) : (subArea +
           ↪ nextCoef));
```

```
165         if (rightChild) total = (total + (subArea + nextCoef));
            subArea = (rightChild ? (subArea - nextCoef) : (subArea +
               ↪ nextCoef));
```

```
188         if (rightChild) total = (total + (subArea + nextCoef));
            subArea = (rightChild ? (subArea - nextCoef) : (subArea +
               ↪ nextCoef));
```

```
229 self.activeLeaf = (up ? (self.activeLeaf + 1) : (self.activeLeaf -
       ↪ 1));
```

```
302         bool leftChild = ((vars.node & 1) == 0);
```

```
318             bool leftChild = ((vars.node & 1) == 0);
```

```
339         bool leftChild = ((vars.node & 1) == 0);
```

CVF-72 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Recommendation Brackets are redundant.

```
49 bool leftChild = ((node & 1) == 0);
```

```
65     bool leftChild = ((node & 1) == 0);
```

```
86 bool leftChild = ((node & 1) == 0);
```

CVF-73 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets are redundant.

```
224 int256 t2 = (idx + 1);
```

```
286     localState.leftChild = ((localState.node & 1) == 0);
```

```
307         localState.leftChild = ((localState.node & 1) == 0);
```

```
331     localState.leftChild = ((localState.node & 1) == 0);
```

```
363 bool branchUp = ((node & 1) != 0);
```

```
404     if (branchUp) total = (total + (subArea + nextCoef));
```

```
448 bool branchUp = ((node & 1) != 0);
```

CVF-74 INFO

- **Category** Procedural
- **Source** DailyJumps.sol

Recommendation Brackets are redundant.

```
33 self.latest = (self.latest + amount);
```

```
41 int256 idx = (day & 1);
```

```
62         self.latest = (self.latest + (self.jumps[0] + self.jumps[1])
             ↪ );
```

```
76 int256 idx = (day & 1);
```

```
82 int256 idx = (day & 1);
```

CVF-75 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Brackets are redundant.

```
114 if (rightChild) total = (total + (subArea + nextCoef));
```

```
116 subArea = (rightChild ? (subArea - nextCoef) : (subArea + nextCoef))
        ↪ ;
    subArea = (subArea / POSITIVE_TWO);
```

CVF-76 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Brackets are redundant.

```
204 self.latest = (self.latest + amount);
```

CVF-77 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Brackets are redundant.

```
75    node = (node >> 1);
```

```
157         gap.stop = (pool.tickBin + 1);
```

```
169 int256 start = (length + startIdx);
```

```
171 int256 stop = (length + stopIdx);
```

```
174     int256 temp = (start & 1);
```

```
178         start = (start + 1);
```

```
182         stop = (stop - 1);
```

```
186     start = (start >> 1);
        stop = (stop >> 1);
```

```
269     node = (node >> 1);
```

CVF-78 INFO

- **Category** Procedural
- **Source** PoolHelper.sol

Recommendation Brackets are redundant.

```
146 int256 mod = (x % y);
```

```
154 int256 r = (x / y);
```

CVF-79 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Brackets are redundant.

```
211 vars.res1 = (vars.ratio0 * vars.reflator);
    vars.res2 = (vars.ratio1 * vars.reflator);
```

CVF-80 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around "0" are redundant.

```
56 if (self.tickBin <= (0)) revert InvalidTickBin(self.tickBin);
```

CVF-81 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Brackets around "depth < scale" and "depth + 1" are redundant.

```
126 for (int256 depth = int256(MIN_SPLITS); (depth < scale); depth = (
    ↪ depth + 1)) {
```

```
141 for (int256 depth = int256(MIN_SPLITS); (depth < scale); depth = (
    ↪ depth + 1)) {
```

```
160     for (int256 depth = int256(MIN_SPLITS); (depth < scale); depth =
        ↪  (depth + 1)) {
```

```
183     for (int256 depth = int256(MIN_SPLITS); (depth < scale); depth =
        ↪  (depth + 1)) {
```

CVF-82 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Brackets around ”depth < scale” and around ”depth + 1” are redundant.

```
109 for (int256 depth = MIN_SPLITS.toInt256(); (depth < scale); depth =
      ↪ (depth + 1)) {
```

CVF-83 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Brackets around ”i < 9” and around ”i + 1” are redundant.

```
36 for (int256 i = 0; (i < 9); i = (i + 1)) {
```

CVF-84 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Brackets around ”idx < 9” and around ”idx + 1” are redundant.

```
46 for (int256 idx = 0; (idx < 9); idx = (idx + 1)) {
```

CVF-85 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Brackets around ”JUMPS - 1” are redundant.

```
274 vars.tailEra = deadEra(pool.era, (JUMPS - 1));
```

```
338 vars.tailEra = deadEra(pool.era, (JUMPS - 1));
```

CVF-86 INFO

- **Category** Procedural
- **Source** PoolHelper.sol

Recommendation Brackets around ”mod + y” are redundant.

```
148 if ((mod ^ y) < 0 && mod != 0) mod = (mod + y);
```

CVF-87 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Brackets around ”offset + 1” are redundant.

```
97 int256 startIdx = self.ofBelow ? (offset + 1) : int256(0);
```

```
113 int256 startIdx = self.ofBelow ? (offset + 1) : int256(0);
```

CVF-88 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Brackets around ”oldIndex < JUMPS” and ”oldIndex + 1” are redundant.

```
86 for (int8 oldIndex = self.nextJump; (oldIndex < JUMPS); oldIndex = (
   ↪ oldIndex + 1)) {
```

CVF-89 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Brackets around ”pastJump - 1” are redundant.

```
323 else if (((pastEra & (1 << (pastJump.toUint256() - 1)).toInt256())
   ↪ != 0)) eraStep = (pastJump - 1);
```

CVF-90 INFO

- **Category** Procedural
- **Source** PoolHelper.sol

Recommendation Brackets around ”r * y != x” and around ”r - 1” are redundant.

```
156 if ((x ^ y) < 0 && (r * y != x)) r = (r - 1);
```

CVF-91 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Brackets around addition are redundant.

```
34 int256 idx = floorMod((bucket + 1), 9);
```

CVF-92 INFO

- **Category** Procedural
- **Source** PoolHelper.sol

Recommendation Brackets around bitwise xor are redundant.

```
148 if ((mod ^ y) < 0 && mod != 0) mod = (mod + y);
```

```
156 if ((x ^ y) < 0 && (r * y != x)) r = (r - 1);
```

CVF-93 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets around comparison are redundant.

```
467 if (self.eraFaberTotals.below != ((newEnd < self.end))) self.halfsum
    ↪  = self.halfsum + diff;
```

```
517 if (self.eraFaberTotals.below != ((newEnd < anchor.end))) anchor.
    ↪ halfsum = anchor.halfsum + diff;
```

CVF-94 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets around condition are redundant.

```
280 for (localState.index = 0; (localState.index < areas.length);
    ↪ localState.index++) {
```

CVF-95 INFO

- **Category** Procedural
- **Source** DailyJumps.sol

Recommendation Brackets around condition are redundant.

```
72 if ((day <= eraDay(pool.era))) revert InvalidDay();
```

CVF-96 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Brackets around condition are redundant.

```
80 if ((scale <= MIN_SPLITS.toInt256())) {
```

CVF-97 INFO

- **Category** Procedural
- **Source** EachPayoff.sol

Recommendation Brackets around condition are redundant.

```
59 Quad t2 = ((side == Z) ? log2Pivot.neg() : log2Pivot);
```

CVF-98 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around division are redundant.

```
117 vars.sweepInput = (pool.epsilon * vars.liquid) / sqrtStrike(pool.
    ↪ splits, pool.tickBin);
```

CVF-99 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around division are redundant.

```
147 vars.transfer = UserPay.Info(vars.consumed ? params.push : params.
    ↪ push - (vars.netInput / exFee(pool.fee)), vars.output.neg());
```

CVF-100 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around division are redundant.

```
175 vars.output = vars.output + (vars.usedInput / mid(pool.splits, pool.
    ↪ tickBin));
```

CVF-101 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around division are redundant.

```
199 vars.transfer = UserPay.Info(vars.output.neg(), vars.consumed ?
    ↪ params.push : params.push - (vars.netInput / exFee(pool.fee)))
    ↪ ;
```

CVF-102 INFO

- **Category** Procedural
- **Source** DeadlineHelper.sol

Recommendation Brackets around expression are redundant.

```
55 if (jumpIndex == 0) return (baseEra + 1);
```

CVF-103 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Recommendation Brackets around expressions are redundant.

```
44 for (int256 index = 0; (index.toUint256() < areas.length); index = (
   ↪ index + 1)) {
```

CVF-104 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Brackets around expressions are redundant.

```
50 for (int256 passed = (self.lastBucket + 1); (passed < bucket);
   ↪ passed = (passed + 1)) {
```

CVF-105 INFO

- **Category** Procedural
- **Source** DeadlineFlag.sol

Recommendation Brackets around expressions are redundant.

```
86 for (int8 oldIndex = self.nextJump; (oldIndex < JUMPS); oldIndex = (
   ↪ oldIndex + 1)) {
```

CVF-106 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around expressions are redundant.

```
39 self.tickBin = (self.tickBin + 1);
```

```
42 self.tickBin = (self.tickBin - 1);
```

CVF-107 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Brackets around expressions are redundant.

```
235 return ((startTub << subSplits(splits).toUint256()), (stopTub <<
    ↪ subSplits(splits).toUint256()));
```

```
248     if ((stopBin <= pool.tickBin)) {
```

```
250         reserve1 = (tubLowSqrt(stopTub) - tubLowSqrt(startTub));
```

```
277 for (int256 i = 0; (i < pool.flowHat.length.toInt256()); i = (i + 1)
    ↪ ) {
```

```
299     vars.yieldRatio = (vars.yieldFlow0 * sqrtStrike(pool.splits,
        ↪ pool.tickBin));
```

CVF-108 INFO

- **Category** Procedural
- **Source** EachPayoff.sol

Recommendation Brackets around function argument expression are redundant.

```
64 return run.read((modulo.toInt128() - WORD_SIZE));
```

CVF-109 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets around function call are redundant.

```
102 if (!(validDeadline(poolEra, deadEra))) revert InvalidDeadline();
```

CVF-110 INFO

- **Category** Procedural
- **Source** DeadlineFlag.sol

Recommendation Brackets around function call are redundant.

```
53 if (!(validDeadline(poolEra, deadEra))) revert InvalidDeadline();
```

CVF-111 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Brackets around function call are redundant.

```
49 if (!(validDeadline(poolEra, deadEra))) revert InvalidDeadline();
```

```
151 if (!(oldDeadEra.isDefined())) revert DeadlineNotDefined();
```

CVF-112 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Brackets around loop condition and around ”index + 1” are redundant.

```
297 for (int256 index = 0; (index.toUint256() < areas.length); index = (
    ↪ index + 1)) {
```

CVF-113 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Brackets around multiplication are redundant.

```
32 int256 bucket = floor((pool.date * POSITIVE_EIGHT));
   Quad fraction = ((pool.date * POSITIVE_EIGHT) - fromInt256(bucket));
```

```
39 return (_sum - (self.ring[uint256(idx)] * fraction));
```

```
43 int256 bucket = floor((pool.date * POSITIVE_EIGHT));
```

CVF-114 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Brackets around multiplication are redundant.

```
210 flowHat0.lateCreate(pool.era, (yieldFlow0.neg() * staged), pool.
       ↪ splits, bin, wrap((DEADERA_NONE)));
    flowHat1.lateCreate(pool.era, (yieldFlow1.neg() * staged), pool.
       ↪ splits, bin, wrap((DEADERA_NONE)));
    pool.owed.createOne(pool.era, pool.splits, (yieldRatio * staged),
       ↪ bin, wrap((DEADERA_NONE)));
```

```
249 flowHat0.lateExpire(pool.era, (params.yieldFlow0 * staged), pool.
       ↪ splits, params.bin, deadEra);
250 flowHat1.lateExpire(pool.era, (params.yieldFlow1 * staged), pool.
       ↪ splits, params.bin, deadEra);
    pool.owed.expireOne(pool.era, pool.splits, (params.yieldRatio.neg()
       ↪ * staged), params.bin, deadEra);
```

CVF-115 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around multiplication are redundant.

```
124 vars.output = vars.output + (vars.usedInput * mid(pool.splits, pool.
       ↪ tickBin));
```

CVF-116 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around multiplication are redundant.

```
168 vars.sweepInput = (pool.epsilon * vars.liquid) * sqrtStrike(pool.
      ↪ splits, pool.tickBin);
```

CVF-117 INFO

- **Category** Procedural
- **Source** NewLoan.sol

Recommendation Brackets around multiplication are redundant.

```
63 return wrap(bytes16(PERIODIC_APPROX.slice((num * 32), 16)));
```

CVF-118 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Brackets around multiplications and divisions are redundant.

```
201 (QuadPacker.repack(FloatBits._apply(exponent, vars.newLower0 - vars.
      ↪ oldLower0)) / vars.lowerSqrtPrice)
     - (QuadPacker.repack(FloatBits._apply(exponent, vars.newUpper0 -
        ↪  vars.oldUpper0)) / vars.upperSqrtPrice)
```

```
205 (QuadPacker.repack(FloatBits._apply(exponent, vars.newUpper1 - vars.
      ↪ oldUpper1)) * vars.upperSqrtPrice)
     - (QuadPacker.repack(FloatBits._apply(exponent, vars.newLower1 -
        ↪  vars.oldLower1)) * vars.lowerSqrtPrice)
```

CVF-119 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Brackets around subtraction are redundant.

```
227 int256 oldNode = int256(self.activeLeaf >> (pool.splits - int256(
    ↪ MIN_SPLITS)).toUint256());
```

```
230 int256 newNode = int256(self.activeLeaf >> (pool.splits - int256(
    ↪ MIN_SPLITS)).toUint256());
```

CVF-120 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets around subtraction are redundant.

```
480 else _start = SignedMath.max((self.end - start), 0);
```

```
483 if (self.eraFaberTtotals.below) _end = SignedMath.min((self.end -
    ↪ start), amounts.length.toInt256());
```

```
499     else _start = SignedMath.max((end - start), 0);
```

```
502     if (self.eraFaberTotals.below) _end = SignedMath.min((end -
        ↪ start), amounts.length.toInt256());
```

CVF-121 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Brackets around subtraction are redundant.

```
152 if (pool.tickBin == (BINS(pool.splits) - 1)) {
```

CVF-122 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Brackets in conditions are redundant.

```
61 while ((start != stop)) {
       if (((start & 1) != 0)) {
```

```
67     if (((stop & 1) != 0)) {
```

CVF-123 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Brackets in expression are redundant.

```
81 if (((start & 1) != 0)) {
```

```
86 if (((stop & 1) != 0)) {
```

CVF-124 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets in expressions are redundant.

```
535 if (((start & 1) != 0)) {
```

```
540 if (((stop & 1) != 0)) {
```

CVF-125 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Recommendation Brackets in the condition are redundant.

```
101 if (((node & 1) != 0)) change = (change + carry[depth.toUint256()]);
```

CVF-126 INFO

- **Category** Procedural
- **Source** DailyJumps.sol

Recommendation Brackets in the expression are redundant.

```
38 if (((day <= self.atDay) && self.touched)) revert InvalidDay();
```

```
40 if (self.touched && (self.atDay >= ((day - 2)))) {
```

```
53     if ((temp == 0)) {
```

```
56         if ((temp == 1)) {
```

```
74 if (!(self.touched)) {
```

CVF-127 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Brackets in the expression are redundant.

```
73  while ((node > 1)) {
```

```
87      if ((temp == 1)) {
```

```
97          if ((self.gapDay != neverAgo)) {
```

```
129 if ((startBin >= stopBin)) revert InvalidStageArguments();
```

```
138 if ((startBin >= pool.tickBin)) {
```

```
140     if ((startBin == pool.tickBin)) startStaged = (pool.tickBin + 1)
        ↪ ;
```

```
146     if ((stopBin <= (pool.tickBin + 1))) {
```

```
148         if ((stopBin == (pool.tickBin + 1))) stopStaged = pool.
            ↪ tickBin;
```

```
173 while ((start != stop)) {
```

```
176     if ((temp != 0)) {
```

```
181     if (((stop & 1) != 0)) {
```

```
216     if ((bin == self.latest.stop)) self.latest.stop = (bin + 1);
        else if ((bin == (self.latest.start - 1))) self.latest.start =
        ↪ bin;
```

```
220 for (int256 i = 0; (i < 2); i = (i + 1)) {
```

```
240 if (((params.bin < gap.start) || (params.bin >= gap.stop))) {
```

```
255     if ((params.bin == gap.stop)) gap.stop = (params.bin + 1);
        else if ((params.bin == (gap.start - 1))) gap.start = params.bin
        ↪ ;
```

```
267 while ((node > 1)) {
```

CVF-128 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Brackets in the expressions are redundant.

```
147 for (int8 oldIndex = self.nextJump; (oldIndex < JUMPS); oldIndex = (
    ↪ oldIndex + 1)) {
```

CVF-129 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Brackets in the expressions are redundant.

```
268 if (((((startTub < 0) || (startTub >= stopTub)) || (stopTub > TUBS))
    ↪  || (liquidity <= POSITIVE_ZERO))) revert InvalidPourArguments
    ↪ ();
```

```
291 if (((vars.startBin <= pool.tickBin) && (pool.tickBin < vars.stopBin
    ↪ ))) {
```

```
293     vars.yieldFlow0 = ((vars.tailYield0 / pool.epsilon) * liquidity)
        ↪ ;
```

```
295     vars.yieldFlow1 = ((vars.tailYield1 / pool.epsilon) * liquidity)
        ↪ ;
```

```
302     vars.yield0 = (vars.yield0 - (vars.tailYield0 * pool.binFrac));
```

```
304     temp0.halfsum = (temp0.halfsum + (vars.yieldFlow0 * pool.binFrac
        ↪ ));
        vars.yield1 = (vars.yield1 + (vars.tailYield1 * pool.binFrac));
```

```
307     temp1.halfsum = (temp1.halfsum - (vars.yieldFlow1 * pool.binFrac
        ↪ ));
```

```
328 if (((((startTub < 0) || (startTub >= stopTub)) || (stopTub > TUBS))
    ↪  || (liquidity <= POSITIVE_ZERO))) revert InvalidPourArguments
    ↪ ();
```

CVF-130 INFO

- **Category** Procedural
- **Source** EachPayoff.sol

Recommendation Brackets in the function argument expression are redundant.

```
60 int256 modulo = ceil(((t2 - pool.date) + HALF));
```

CVF-131 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Brackets in the right part of the assignment are redundant.

```
140 if ((startBin == pool.tickBin)) startStaged = (pool.tickBin + 1);
```

```
216 if ((bin == self.latest.stop)) self.latest.stop = (bin + 1);
```

CVF-132 INFO

- **Category** Procedural
- **Source** DeadlineHelper.sol

Recommendation Brackets within expression are redundant.

```
34 return jumpBitLength(((deadline - fromEra) - 1));
```

CVF-133 INFO

- **Category** Documentation
- **Source** Constants.sol

Recommendation Consider explaining how these values were calculated.

```
27 //precalulated values
```

CVF-134 INFO

- **Category** Suboptimal
- **Source** InfinityPool.sol

Recommendation Consider implementing a separate view-only version of this function, as off-chain execution of a state-modifying function seems weird.

```
183 *  To save gas, one should call this function using callStatic.
```

CVF-135 INFO

- **Category** Suboptimal
- **Source** Structs.sol

Recommendation Consider turning these constants into a enum.

```
7 int8 constant LP_STATE_OPENED = 1; // not earning yet
  int8 constant LP_STATE_ACTIVE = 2; // earning
  int8 constant LP_STATE_CLOSED = 3;
```

CVF-136 INFO

- **Category** Suboptimal
- **Source** EraFaberTotals.sol

Recommendation Consider using bitwise operations instead of the "%" operator.

```
100 if (((stopIdx % (1 << (scale - MIN_SPLITS.toInt256()).toUint256()).
    ↪ toInt256()) == 0)) {
```

CVF-137 INFO

- **Category** Suboptimal
- **Source** Spot.sol

Recommendation Conversion from ”int256” is redundant, as Solidity compiler would do it automatically anyway.

```
25 error InvalidPushAmount();
```

```
28     if (isUp) self.joinStaged.flush(self, int256(int24(self.tickBin
          ↪ + 1)));
       else self.joinStaged.flush(self, int256(int24(self.tickBin - 1))
          ↪ );
```

CVF-138 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation Conversion to ”int8” is redundant, as ”self.nextJump” is already ”int8”.

```
187 while (self.nextJump < JUMPS && (poolEra >= (self.deadEra(int8(self.
       ↪ nextJump))))) {
```

CVF-139 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation Conversion to ”int8” is redundant, as ”t1” is already ”int8”.

```
299 int256 lastJump = (t1 < (JUMPS - 1)) ? int8(t1) : (JUMPS - 1);
```

CVF-140 INFO

- **Category** Suboptimal
- **Source** Swapper.sol

Recommendation Conversions to ”int8” are redundant, as ”self.superC.nextJump” is already ”int8”.

```
34 while ((self.superC.nextJump < JUMPS) && (pool.era >= (self.superC.
      ↪ deadEra(int8(self.superC.nextJump))))) {
       Quad nextDate = eraDate(self.superC.deadEra(int8(self.superC.
          ↪ nextJump)));
```

```
39     uint8 idx = uint8(self.superC.nextJump);
```

CVF-141 INFO

- **Category** Readability
- **Source** IInfinityPoolState.sol

Description Declaring a top-level structure in a file named after an interface makes it harder navigating through code.

Recommendation Consider either moving the structure declaration into the interface or moving it into a separate file.

```
26 struct PoolState {
```

CVF-142 INFO

- **Category** Suboptimal
- **Source** GrowthSplitFrame.sol

Description Excessive use of "toUint256" seems ugly and inefficient.

Recommendation Consider changing the type of corresponding arguments, fields and variables to "uint256" to avoid conversions.

```
36  self.activeLeaf = int256(1 << pool.splits.toUint256()) + initBin;
```

```
41  return self.large[self.activeLeaf.toUint256() >> (splits - int256(
        ↪ MIN_SPLITS)).toUint256()];
```

```
45  return self.large[node.toUint256()].grow(poolEra, poolDeflator);
```

```
49  return self.large[node.toUint256()].tail;
```

```
53  return self.large[node.toUint256()].deadlineJumps.now(pool.era);
```

```
65  if (!(((scale <= int256(MIN_SPLITS) && (0 <= startIdx)) && (stopIdx
        ↪ <= int256(1 << scale.toUint256())))))) revert
        ↪ InvalidSumTotalArguments();
```

```
68  int256 start = (int256(1 << scale.toUint256()) + startIdx);
    int256 stop = (int256(1 << scale.toUint256()) + stopIdx);
```

```
101 int256 activeTub = (int256(self.activeLeaf - int256(1 << pool.splits
        ↪ .toUint256())) >> (pool.splits - int256(MIN_SPLITS)).toUint256
        ↪ ());
```

CVF-143 INFO

- **Category** Readability
- **Source** DeadlineFlag.sol

Description Here "drop" is guaranteed to be true.

Recommendation Consider using "true".

```
93  dropoff(self, drop);
```

```
104 self.drops[int256(newIndex).toUint256()] = drop;
```

CVF-144 INFO

- **Category** Suboptimal
- **Source** DeadlineHelper.sol

Recommendation Here “require” should be used instead of “assert”.

```
54 assert((jumpIndex >= 0) && (jumpIndex < JUMPS));
```

CVF-145 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Inner brackets are redundant.

```
104 if (!(self.touched)) {
```

```
118 if ((jump < self.nextJump)) self.nextJump = jump;
```

```
154         if ((deadline <= poolEra)) {
```

```
165
```

CVF-146 INFO

- **Category** Procedural
- **Source** Payoff.sol

Recommendation Inner brackets are redundant.

```
23 if ((token == Z)) {
```

```
38 if ((token == Z)) {
```

CVF-147 INFO

- **Category** Procedural
- **Source** DailyJumps.sol

Recommendation Inner brackets are redundant.

```
28 if (!(self.touched)) {
```

CVF-148 INFO

- **Category** Procedural
- **Source** DeadlineFlag.sol

Recommendation Inner brackets are redundant.

```
55 if (!(self.touched)) {
69 if ((jump < self.nextJump)) self.nextJump = jump;
89     if ((drop != false)) {
92         if ((deadline <= poolEra)) {
102             if ((newIndex == oldIndex)) break;
```

CVF-149 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Inner brackets are redundant.

```
51 if (!(self.touched)) {
```

CVF-150 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Inner brackets are redundant.

```
65  if ((jump < self.nextJump)) self.nextJump = jump;
```

```
89      if ((drop != POSITIVE_ZERO)) {
```

```
92          if ((deadline <= poolEra)) {
```

```
102             if ((newIndex == oldIndex)) break;
```

```
198 if (!(self.touched)) {
```

```
312             if ((drop != POSITIVE_ZERO)) {
                    if ((pastJump == -1)) {
```

```
322                         if ((pastJump == 0)) eraStep = 0;
```

CVF-151 INFO

- **Category** Procedural
- **Source** DeadlineHelper.sol

Recommendation Inner brackets are redundant.

```
54  assert((jumpIndex >= 0) && (jumpIndex < JUMPS));
```

CVF-152 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
75  function dropOff(AnchorSet.Info storage self, Anchor.Info storage
    ↪ drop) internal {}
```

CVF-153 INFO

- **Category** Procedural
- **Source** DeadlineFlag.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
29 function dropOff(DeadlineFlag.Info storage self, bool drop) internal
   ↪  {}
```

CVF-154 INFO

- **Category** Procedural
- **Source** IInfinityPoolState.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
24 interface IInfinityPoolState {}
```

CVF-155 INFO

- **Category** Documentation
- **Source** Constants.sol

Description It is unclear what this value actually is.

Recommendation Consider documenting.

```
61 Quad constant LOG1PPC = Quad.wrap(0x3ff8460d6ccca3676b71e159f1d244a4
   ↪ );
```

CVF-156 INFO

- **Category** Suboptimal
- **Source** BoxcarTubFrame.sol

Recommendation It would be more efficient to use the “addRange” function here.

```
79 add(self, int256(MIN_SPLITS), startTub, stopTub, change);
```

CVF-157 INFO

- **Category** Procedural
- **Source** NewLoan.sol

Recommendation One of these duplicated imports should be removed.

```
22 import {PoolState} from "../../interfaces/IInfinityPoolState.sol";
```

```
30 import {PoolState} from "../../interfaces/IInfinityPoolState.sol";
```

CVF-158 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Outer brackets are redundant.

```
39 int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

```
71     return (trunkValue + leaf.deadlineJumps.now(poolEra));
```

```
78 int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

```
117 int256 cursor = ((1 << splits.toUint256()).toInt256() + offset);
```

CVF-159 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Outer brackets are redundant.

```
68  int256 start = (int256(1 << scale.toUint256()) + startIdx);
    int256 stop = (int256(1 << scale.toUint256()) + stopIdx);
```

```
72      if (((start & 1) != 0)) {
            total = (total + largeAt(self, poolEra, poolDeflator, start)
               ↪ );
            start = (start + 1);
```

```
76      if (((stop & 1) != 0)) {
            stop = (stop - 1);
            total = (total + largeAt(self, poolEra, poolDeflator, stop))
               ↪ ;
```

```
80      start = (start >> 1);
        stop = (stop >> 1);
```

```
122 int256 node = (int256(1 << scale.toUint256()) + offset);
    int256 temp = ((scale > int256(MIN_SPLITS)) ? int256(node >> (scale
       ↪ - int256(MIN_SPLITS)).toUint256()) : node);
```

```
128     bool rightChild = ((offset & int256(1 << ((scale - depth).
           ↪ toUint256() - 1))) != 0);
```

```
130     subArea = (subArea / POSITIVE_TWO);
```

```
137 int256 node = (int256(1 << scale.toUint256()) + offset);
    int256 temp = ((scale > int256(MIN_SPLITS)) ? int256(node >> (scale
       ↪ - int256(MIN_SPLITS)).toUint256()) : node);
```

```
143     bool rightChild = ((offset & int256(1 << ((scale - depth).
           ↪ toUint256() - 1))) != 0);
        subArea = (rightChild ? (subArea - nextCoef) : (subArea +
           ↪ nextCoef));
        subArea = (subArea / POSITIVE_TWO);
```

```
151 if (((stopIdx % int256(1 << (scale - int256(MIN_SPLITS)).toUint256()
       ↪ )) == 0)) {
```

```
167         subArea = (subArea / POSITIVE_TWO);
```

```
169     return (total / POSITIVE_TWO);
```



```
174 if (((stopIdx % int256(1 << (scale - int256(MIN_SPLITS)).toUint256()
       ↪ )) == 0)) {
```

CVF-160 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Recommendation Outer brackets are redundant.

```
45      node = (first + index);
```

```
47      depth = (scale - 1);
        while ((depth.toUint256() >= MIN_SPLITS)) {
```

```
51          node = (node >> 1);
```

```
57              change = (change + carry[depth.toUint256()]);
```

```
59          depth = (depth - 1);
```

```
62      if ((depth.toUint256() < MIN_SPLITS)) {
```

```
67              node = (node >> 1);
```

```
72                  change = (change + carry[depth.toUint256()]);
                    if ((node <= 1)) break;
```

```
77              depth = (depth - 1);
```

```
82  if ((depth.toUint256() >= MIN_SPLITS)) {
```

```
84      while ((depth.toUint256() > MIN_SPLITS)) {
            depth = (depth - 1);
```

```
88          node = (node >> 1);
```

```
93              change = (change + carry[depth.toUint256()]);
```

```
98  while ((node > 1)) {
```

```
100     depth = (depth - 1);
```

```
103     node = (node >> 1);
```

CVF-161 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Outer brackets are redundant.

```
32 int256 node = (int256(1 << scale.toUint256()) + offset);
   while ((node > 1)) {
       total = (total + coefAt(self, poolEra, node));
       node = (node >> 1);
```

CVF-162 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Outer brackets are redundant.

```
78 int256 start = (length + startIdx);
   int256 stop = (length + stopIdx);
80 while ((start != stop)) {
```

```
83         start = (start + 1);
```

```
87         stop = (stop - 1);
```

```
91     start = (start >> 1);
       stop = (stop >> 1);
```

CVF-163 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Outer brackets are redundant.

```
57 else shouldAdd = (bin >= validEnd(self,
      ↪ jumpyAnchorFaberDataStructure));
```

CVF-164 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Outer brackets are redundant.

```
278 localState.first = ((1 << scale.toUint256()).toInt256() + start);
```

```
281     localState.node = (localState.first + int256(localState.index));
```

```
283     localState.depth = (scale - 1);
```

```
288         localState.node = (localState.node >> 1);
```

```
295                 self, poolEra, localState.node, (localState.carry[
                        ↪ localState.depth.toUint256()] - localState.
                        ↪ change), deadEra
```

```
297             localState.change = (localState.change + localState.
                  ↪ carry[localState.depth.toUint256()]);
```

```
300         localState.depth = (localState.depth - 1);
```

```
308             localState.node = (localState.node >> 1);
```

```
314                 localState.change = (localState.change + localState.
                      ↪ carry[localState.depth.toUint256()]);
```

```
321             localState.depth = (localState.depth - 1);
```

```
330         localState.depth = (localState.depth - 1);
```

```
332         localState.node = (localState.node >> 1);
```

```
338                 self, poolEra, localState.node, (localState.carry[
                        ↪ localState.depth.toUint256()] - localState.
                        ↪ change), deadEra
```

```
340             localState.change = (localState.change + localState.
                  ↪ carry[localState.depth.toUint256()]);
```

```
347     localState.depth = (localState.depth - 1);
```

```
349     if ((localState.node & 1) != 0) localState.change = (localState.
          ↪ change + localState.carry[localState.depth.toUint256()]);
```

```
351 localState.node = (localState.node >> 1);
```

```
360 int256 node = ((1 << scale.toUint256()).toInt256() + offset);
```

CVF-165 INFO

- **Category** Procedural
- **Source** BoxcarTubFrame.sol

Recommendation Outer brackets are redundant.

```
32 if (((offset < 0) || (offset >= int256(1 << scale.toUint256())))
      ↪ revert OffsetOutOfRange();

34 int256 node = (int256(1 << scale.toUint256()) + offset);
   while ((node > 1)) {
       total = (total + coefAt(self, node));
       node = (node >> 1);

56 if ((((startIdx < 0) || (startIdx > stopIdx)) || (stopIdx > length))
      ↪ ) revert StartIdOrStopIdOutOfRange();
   int256 start = (length + startIdx);

59 int256 stop = (length + stopIdx);
60 while ((start != stop)) {
       if (((start & 1) != 0)) {

63         start = (start + 1);

65     if (((stop & 1) != 0)) {
           stop = (stop - 1);

69     start = (start >> 1);
70     stop = (stop >> 1);
```

CVF-166 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Outer brackets are redundant.

```
33 Quad fraction = ((pool.date * POSITIVE_EIGHT) - fromInt256(bucket));

37     _sum = (_sum + self.ring[uint256(i)]);

39 return (_sum - (self.ring[uint256(idx)] * fraction));
```

CVF-167 INFO

- **Category** Procedural
- **Source** DailyJumps.sol

Recommendation Outer brackets are redundant.

```
51 int256 temp = (eraDay(pool.era) - self.atDay);
```

```
57         int256 idx = (eraDay(pool.era) & 1);
```

```
59         self.latest = (self.latest + self.jumps[idx.toUint256()]);
```

```
84 self.jumps[idx.toUint256()] = (self.jumps[idx.toUint256()] + amount)
   ↪ ;
```

CVF-168 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Outer brackets are redundant.

```
57 int256 start = ((1 << scale.toUint256()).toInt256() + startIdx);
```

```
59 int256 stop = ((1 << scale.toUint256()).toInt256() + stopIdx);
```

```
63         total = (total + largeAtDropReader(self, start, deadEra));
           start = (start + 1);
```

```
68         stop = (stop - 1);
           total = (total + largeAtDropReader(self, stop, deadEra));
```

```
72     start = (start >> 1);
       stop = (stop >> 1);
```

```
90         (startIdx >> (scale - MIN_SPLITS.toInt256()).toUint256()),
           (stopIdx >> (scale - MIN_SPLITS.toInt256()).toUint256()),
```

CVF-169 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Outer brackets are redundant.

```
112 bool rightChild = ((stopIdx & (1 << ((scale - depth) - 1).toUint256
       ↪ ()).toInt256()) != 0);
```

CVF-170 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Outer brackets are redundant.

```
326         pastEra = (pastEra + (1 << eraStep.toUint256()).toInt256());
            nextDeflator = (nextDeflator * wrap(bytes16(DEFLATOR_STEPS.
               ↪ slice((eraStep * 32).toUint256(), 16))));
            pastJump = (pastJump + 1);
```

```
332 self.deadlineJumps.latest = (self.deadlineJumps.latest - drop);
```

CVF-171 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Outer brackets are redundant.

```
71  int256 node = ((1 << scale.toUint256()).toInt256() + offset);
```

```
74      total = (total + nowReader_coefAt(self, pool, node));
```

```
82  int256 temp = (eraDay(pool.era) - self.gapDay);
```

```
88          int256 idx1 = (eraDay(pool.era) & 1);
```

```
123 int256 idx = ((1 << pool.splits.toUint256()).toInt256() + bin);
```

```
131 int256 tailDay = (eraDay(pool.era) + 2);
```

```
134 int256 idx = (eraDay(pool.era) & 1);
```

```
197 Quad yieldFlow0 = (fees0.tailAt(pool, bin) / pool.epsilon);
```

```
199 Quad yieldFlow1 = (fees1.tailAt(pool, bin) / pool.epsilon);
```

```
201 Quad yieldRatio = (yieldFlow0 * sqrtStrike(pool.splits, bin));
```

```
265 int256 node = ((1 << scale.toUint256()).toInt256() + offset);
```

```
268     total = (total + liftReader_coefAt(self, node, day));
```

CVF-172 INFO

- **Category** Procedural
- **Source** EachPayoff.sol

Recommendation Outer brackets are redundant.

```
57 Quad t1 = (sqRoot ? (logTick() / POSITIVE_TWO) : logTick());
   Quad log2Pivot = ((t1 * fromInt256(pivotTick)) / LAMBDA);
   Quad t2 = ((side == Z) ? log2Pivot.neg() : log2Pivot);
```

```
63 SparseFloat.Info storage run = (sqRoot ? pool.reserveRun[t3] : pool.
   ↪ priceRun[t3]);
```

CVF-173 INFO

- **Category** Procedural
- **Source** DeadlineHelper.sol

Recommendation Outer brackets are redundant.

```
56 else return (((baseEra >> (uint256(int256(jumpIndex - 1)))) + 2) <<
   ↪ uint256(int256(jumpIndex - 1)));
```

CVF-174 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Outer brackets in expressions are redundant.

```
349     if ((localState.node & 1) != 0) localState.change = (localState.
           ↪ change + localState.carry[localState.depth.toUint256()]);
```

```
395     Quad subArea = largeAtNowReader(self, poolEra, ((1 << MIN_SPLITS
           ↪ ).toInt256() + (stopIdx >> (scale - MIN_SPLITS.toInt256())
           ↪ .toUint256())));
```

```
399     for (int256 depth = MIN_SPLITS.toInt256(); (depth < scale);
           ↪ depth = (depth + 1)) {
400         Quad nextCoef = smallAtNowReader(self, poolEra, ((1 << depth
               ↪ .toUint256()).toInt256() + (stopIdx >> (scale - depth)
               ↪ .toUint256())));
```

```
423 if ((scale <= MIN_SPLITS.toInt256())) {
```

```
447 while ((node >= (2 << MIN_SPLITS).toInt256())) {
```

```
486 for (int256 index = _start; (index < _end); index = (index + 1)) {
```

```
505     for (int256 index = _start; (index < _end); index = (index + 1))
           ↪  {
```

CVF-175 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Recommendation Outer brackets in the assignment are redundant.

```
101 if (((node & 1) != 0)) change = (change + carry[depth.toUint256()]);
```

CVF-176 INFO

- **Category** Readability
- **Source** InfinityPool.sol

Recommendation Should be “token1” instead of “token0”.

```
102 * @return amount1 The amount of token0 transferred in the swap. The
      ↪ convention is amount > 0 => user pays, amount < 0 => user
      ↪ receives.
```

CVF-177 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Some brackets are redundant.

```
41 while (!(self.tree[cursor.toUint256()].deadlineJumps.touched)) {
```

```
80 while (!(self.tree[cursor.toUint256()].deadlineJumps.touched)) {
```

CVF-178 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Some brackets are redundant.

```
151     int128 exponent = int128(ceil((vars.drainDate.neg() + HALF))) -
           ↪ int128(WORD_SIZE);
```

```
210     vars.reflator = (((LAMBDA * vars.drainDate)).exp() * self.
           ↪ liquidity);
```

```
317     UserPay.Info((liquidity * (vars.reserve0 + (vars.yield0 *
           ↪ tailDeflator))), (liquidity * (vars.reserve1 + (vars.
           ↪ yield1 * tailDeflator))));
```

```
349     UserPay.Info((liquidity * (vars.reserve0 + (vars.yield0 *
           ↪ tailDeflator))), (liquidity * (vars.reserve1 + (vars.
           ↪ yield1 * tailDeflator))));
```

```
353 return (((logBin(pool.splits) / POSITIVE_TWO) * fromInt256((bin -
       ↪ BINS(pool.splits) / (2))))).exp();
```

```
389 vars.exponent = int128(ceil((pool.date.neg() + HALF))) - int128(
       ↪ WORD_SIZE);
```

CVF-179 INFO

- **Category** Procedural
- **Source** Swapper.sol

Recommendation Some brackets are redundant.

```
34    while ((self.superC.nextJump < JUMPS) && (pool.era >= (self.
         ↪ superC.deadEra(int8(self.superC.nextJump))))) {
```

```
37        self.accrued = (self.accrued + (((pool.twapSpread * LN2) *
             ↪ self.superC.latest) * (nextDate - self.atDate)));
```

```
40        self.superC.latest = (self.superC.latest - self.superC.drops
             ↪ [idx]);
```

```
44    self.accrued = (self.accrued + (((pool.twapSpread * LN2) * self.
         ↪ superC.latest) * (pool.date - self.atDate)));
```

```
52 self.latest = (self.latest - drop);
```

```
62 if (!(self.superC.touched)) self.atDate = pool.date;
```

```
106    Quad inflator = (POSITIVE_ONE / pool.deflator);
```

```
204    vars.ratio = (self.tokenMix / vars.capacity);
       res = (fixedToken == I) ? vars.ratio : (POSITIVE_ONE - vars.
          ↪ ratio);
```

```
237            res = self.tokenMix * (POSITIVE_ONE + (pool.twapSpread *
                  ↪  LAMBDA * vars.twapLeft));
```

```
248                    vars.owedMean = ((vars.twapDeflator - vars.
                          ↪ deadDeflator) * pool.epsilon * self.
                          ↪ oweLimit);
```

```
254                res = (token == Z)
                       ? (((pool.deflator - vars.twapDeflator) * pool.
                          ↪ epsilon * self.oweLimit) / sqrtStrike(pool
                          ↪ .splits, self.strikeBin))
```

```
257                    : (((pool.deflator - vars.twapDeflator) * pool.
                          ↪ epsilon * self.oweLimit) * sqrtStrike(pool
                          ↪ .splits, self.strikeBin))
```

```
261                    (token == Z) ? (POSITIVE_ONE / sqrtStrike(pool.
                          ↪ splits, self.strikeBin)) : sqrtStrike(pool
                          ↪ .splits, self.strikeBin);
```



```
263                res = (((pool.epsilon * self.oweLimit) * pool.
                       ↪ deflator) - (self.tokenMix * vars.towardToken)
```

CVF-180 INFO

- **Category** Procedural
- **Source** NewLoan.sol

Recommendation Some brackets are redundant.

```
104 vars.inflator = (POSITIVE_ONE / pool.deflator);
```

```
108 if ((params.lockinEnd <= POSITIVE_ONE)) {
```

```
112     if ((params.lockinEnd > POSITIVE_ZERO)) {
            lockingEndVars.geoMean = ((params.lockinEnd * quadVar(pool))
               ↪ ).sqrt();
            lockingEndVars.shiftMnot1 = (
                periodicApproxConstant(0) + (periodicApproxConstant(1) *
                   ↪  lockingEndVars.geoMean)
                    + (periodicApproxConstant(2) + quadVar(pool) *
                       ↪ periodicApproxConstant(3)) * params.lockinEnd
            ) * lockingEndVars.geoMean;
            lockingEndVars.shiftMis1 = (
                periodicApproxConstant(4) + (periodicApproxConstant(5) *
                   ↪  lockingEndVars.geoMean)
                    + (periodicApproxConstant(6) + quadVar(pool) *
                       ↪ periodicApproxConstant(7)) * params.lockinEnd
            ) * lockingEndVars.geoMean;
            lockingEndVars.logw = max((params.logm + lockingEndVars.
               ↪ shiftMnot1), lockingEndVars.shiftMis1);
            if (!((lockingEndVars.logw > POSITIVE_ZERO))) revert
               ↪ UnavailableLockInPeriod(params.lockinEnd);
```

```
136 params.logm = (params.logm + params.deltaLogm);
    vars.theta = (POSITIVE_ONE + (POSITIVE_EIGHT_OVER_NINE * vars.rate))
       ↪ ;
    vars.cr = (vars.rate / POSITIVE_NINE);
    vars.bin = (params.startBin + int256(params.index));
140 vars.oldUsed = (pool.used.nowAt(pool.era, pool.splits, vars.bin) *
       ↪ pool.deflator);
```

```
142 vars.virtualOwed = (((vars.theta + (vars.cr / (POSITIVE_ONE - vars.
       ↪ oldUsed))) * vars.oldUsed) + (params.owedPotentialAtIndex /
       ↪ vars.minted));
    vars.square = ((vars.virtualOwed - vars.theta) + vars.cr);
    vars.newUsed = (
        (((vars.virtualOwed + POSITIVE_ONE) + vars.rate) - (((vars.
           ↪ square * vars.square) + ((POSITIVE_FOUR * vars.cr) * vars.
           ↪ theta))).sqrt())
            / (POSITIVE_TWO * vars.theta)
    );
    return (((vars.newUsed - vars.oldUsed) * vars.minted) * vars.
       ↪ inflator);
```



```
179 if (!((params.owedPotential.length > 0) && (params.lockinEnd >=
```

CVF-181 INFO

- **Category** Procedural
- **Source** Advance.sol

Recommendation Some brackets are redundant.

```
93  if ((toDate < pool.date)) revert TargetDateInPast();

99  for (; (vars.jumpIndex < JUMPS); vars.jumpIndex = (vars.jumpIndex +
      ↪ 1)) {

104     if ((vars.nextDate > toDate)) break;

115         if (!(((vars.freed0 == POSITIVE_ZERO) && (vars.freed1 ==
              ↪ POSITIVE_ZERO)))) {

120             vars.freed0 = (vars.freed0 - lentEnd0.eraFaberTotals.
                  ↪ dropSum(pool, (pool.tickBin + 1), vars.deadline));
                vars.freed1 = (vars.freed1 - lentEnd1.eraFaberTotals.
                  ↪ dropSum(pool, pool.tickBin, vars.deadline));
                vars.freed0 = (vars.freed0 * pool.deflator);
                vars.freed1 = (vars.freed1 * pool.deflator);
                vars.needed = (pool.lent.dropAt(pool.splits, pool.
                  ↪ tickBin, vars.deadline) * pool.deflator);

127                 (vars.freed0 - ((((POSITIVE_ONE - pool.binFrac) *
                      ↪ pool.epsilon) * vars.needed) / sqrtStrike(pool
                      ↪ .splits, pool.tickBin)));

129             vars.surplus1 = (vars.freed1 - (((pool.binFrac * pool.
                  ↪ epsilon) * vars.needed) * sqrtStrike(pool.splits,
                  ↪ pool.tickBin)));

131             if (((vars.surplus0 >= POSITIVE_ZERO) && (vars.surplus1
                  ↪ >= POSITIVE_ZERO))) {
                    pool.surplus0 = (pool.surplus0 + vars.surplus0);
                    pool.surplus1 = (pool.surplus1 + vars.surplus1);

135                 if (((vars.surplus0 <= POSITIVE_ZERO) && (vars.
                      ↪ surplus1 <= POSITIVE_ZERO))) {
                        vars.hole0 = (
                            (vars.surplus0 == POSITIVE_ZERO)
                                ? POSITIVE_ZERO
                                : (vars.surplus0.neg() / (expire0.dropAt
                                    ↪ (vars.deadline) * pool.deflator))
140                     );

142                     vars.hole1 = (
                            (vars.surplus1 == POSITIVE_ZERO)
                                ? POSITIVE_ZERO
                                : (vars.surplus1.neg() / (expire1.dropAt
```

CVF-182 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(drop.touched && drop.end == 0 && drop.halfsum == POSITIVE_ZERO)" or even as "drop.touched || drop.end == 0 || drop.halfsum == POSITIVE_ZERO".

```
151 if ((!(drop.touched && drop.end == 0 && drop.halfsum ==
    ↪ POSITIVE_ZERO))) {
```

CVF-183 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The condition could be simplified as: pastEra & 1 « pastJump.toUint256() - 1.toInt256() != 0

```
323 else if (((pastEra & (1 << (pastJump.toUint256() - 1)).toInt256())
    ↪ != 0)) eraStep = (pastJump - 1);
```

CVF-184 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(0 <= startIdx && startIdx <= stopIdx && stopIdx <= (1 « scale.toUint256()).toInt256())" or even as "0 > startIdx || startIdx > stopIdx || stopIdx > (1 « scale.toUint256()).toInt256()"

```
83 if (!((((0 <= startIdx) && (startIdx <= stopIdx)) && (stopIdx <= (1
   ↪ << scale.toUint256()).toInt256())))) {
```

CVF-185 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(0 <= startIdx && startIdx <= stopIdx && stopIdx <= (1 « scale.toUint256()).toInt256())" or even as "0 > startIdx || startIdx > stopIdx || stopIdx > (1 « scale.toUint256()).toInt256()".

```
426 if (!((((0 <= startIdx) && (startIdx <= stopIdx)) && (stopIdx <= (1
    ↪ << scale.toUint256()).toInt256())))) {
```

CVF-186 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(0 <= startIdx && startIdx <= stopIdx && stopIdx <= length)" or even as "0 > startIdx || startIdx > stopIdx || stopIdx > length".

```
77 if (!((((0 <= startIdx) && (startIdx <= stopIdx)) && (stopIdx <=
   ↪ length)))) revert InvalidAddRangeArguments();
```

CVF-187 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(0 <= startIdx && stopIdx <= int256(1 « scale.toUint256()))" or even to "0 > startIdx || stopIdx > int256(1 « scale.toUint256())"

```
200 if (!((0 <= startIdx) && (stopIdx <= int256(1 << scale.toUint256())))
    ↪ ) revert InvalidSumRangeArguments();
```

CVF-188 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale <= MIN_SPLITS.toInt256() && 0 <= startIdx && startIdx <= stopIdx && stopIdx <= (1 « scale.toUint256()).toInt256())" or even as "scale > MIN_SPLITS.toInt256() || 0 > startIdx || startIdx > stopIdx || stopIdx > (1 « scale.toUint256()).toInt256()".

```
524 if (!(((scale <= MIN_SPLITS.toInt256() && ((0 <= startIdx)) && (
    ↪ startIdx <= stopIdx)) && (stopIdx <= (1 << scale.toUint256()).
    ↪ toInt256())))) {
```

CVF-189 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale <= MIN_SPLITS.toInt256() && 0 <= startIdx && startIdx <= stopIdx && stopIdx <= (1 « scale.toUint256()).toInt256()))" or even as "scale > MIN_SPLITS.toInt256() || 0 > startIdx || startIdx > stopIdx || stopIdx > (1 « scale.toUint256()).toInt256())"

```
51 if (!(((scale <= MIN_SPLITS.toInt256() && ((0 <= startIdx)) && (
    ↪ startIdx <= stopIdx)) && (stopIdx <= (1 << scale.toUint256()).
    ↪ toInt256())))) {
```

CVF-190 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale >= int256(MIN_SPLITS) && stopIdx & int256(-1 « scale.toUint256()) == 0)" or even as "scale < int256(MIN_SPLITS) || stopIdx & int256(-1 « scale.toUint256()) != 0".

```
177 if (!(scale >= int256(MIN_SPLITS) && (((stopIdx & int256(-1 << scale
    ↪ .toUint256())) == 0)))) revert InvalidSubLeftSumArguments();
```

CVF-191 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale >= MIN_SPLITS.toInt256() && offset & -1 « scale.toUint256() == 0)" or even as "scale < MIN_SPLITS.toInt256() || offset & -1 « scale.toUint256() != 0".

```
358 if (!(scale >= MIN_SPLITS.toInt256() && (((offset & (-1 << scale.
    ↪ toUint256())) == 0)))) revert InvalidAddArguments();
```

CVF-192 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale >= MIN_SPLITS.toInt256() && offset & -1 « scale.toUint256() == 0)" or even as "scale < MIN_SPLITS.toInt256() || offset & -1 « scale.toUint256() != 0".

```
443 if (!(scale >= MIN_SPLITS.toInt256() && (((offset & (-1 << scale.
    ↪ toUint256())) == 0)))) revert InvalidAddArguments();
```

CVF-193 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale >= MIN_SPLITS.toInt256() && start >= 0 && (start + areas.length.toInt256() <= (1 « scale.toUint256()).toInt256()))" or even as "scale < MIN_SPLITS.toInt256() || start < 0 || (start + areas.length.toInt256() > (1 « scale.toUint256()).toInt256())".

```
273 if (!((scale >= MIN_SPLITS.toInt256() && ((start >= 0)) && ((start +
    ↪  areas.length.toInt256()) <= (1 << scale.toUint256()).toInt256
    ↪ ()))))) {
```

CVF-194 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale >= MIN_SPLITS.toInt256() && stopIdx & -1 « scale.toUint256() == 0)" or even as "scale < MIN_SPLITS.toInt256() || stopIdx & -1 « scale.toUint256() != 0"

```
103 if (!(scale >= MIN_SPLITS.toInt256() && (((stopIdx & (-1 << scale.
    ↪ toUint256()))) == 0)))) revert InvalidSubLeftSumArguments();
```

CVF-195 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale >= MIN_SPLITS.toInt256() && stopIdx & -1 « scale.toUint256() == 0)" or even as "scale < MIN_SPLITS.toInt256() || stopIdx & -1 « scale.toUint256() != 0".

```
393 if (!(scale >= MIN_SPLITS.toInt256() && (((stopIdx & (-1 << scale.
    ↪ toUint256()))) == 0)))) revert InvalidSubLeftSumArguments();
```

CVF-196 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale.toUint256() >= MIN_SPLITS && start >= 0 && start.toUint256() + areas.length <= 1 « scale.toUint256())" or even as "scale.toUint256() < MIN_SPLITS || start < 0 || start.toUint256() + areas.length > 1 « scale.toUint256()"

```
287 if (!((scale.toUint256() >= MIN_SPLITS && ((start >= 0)) && ((start.
    ↪ toUint256() + areas.length) <= (1 << scale.toUint256())))))) {
```

CVF-197 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as "!(scale.toUint256() >= MIN_SPLITS && start >= 0 && start.toUint256() + areas.length <= 1 « scale.toUint256())" or even as "scale.toUint256() < MIN_SPLITS || start < 0 || start.toUint256() + areas.length > 1 « scale.toUint256()".

```
30 if (!((scale.toUint256() >= MIN_SPLITS && ((start >= 0)) && ((start.
   ↪ toUint256() + areas.length) <= (1 << scale.toUint256())))) {
```

CVF-198 INFO

- **Category** Procedural
- **Source** DeadlineHelper.sol

Recommendation Some brackets are redundant. The expression could be simplified as "maybeIndex != JUMPS && (maybeIndex <= 1 || deadline & ~(-1 « uint256(int256(maybeIndex - 1))) == 0)" or even "maybeIndex != JUMPS && (maybeIndex <= 1 || deadline « uint256(int256(257 - maybeIndex))) == 0)".

```
41 if ((maybeIndex != JUMPS) && ((maybeIndex <= 1) || ((deadline &
   ↪ ~((-1 << uint256(int256(maybeIndex - 1))))) == 0))) return
   ↪ maybeIndex;
```

CVF-199 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as "offset & int256(-1 « scale.toUint256()) != 0" or even as "uint256(offset) » scale.toUint256() != 0".

```
121 if (!(((offset & int256(-1 << scale.toUint256())) == 0))) revert
    ↪ InvalidAreaArguments();
```

```
136 if (!(((offset & int256(-1 << scale.toUint256())) == 0))) revert
    ↪ InvalidAreaArguments();
```

CVF-200 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Some brackets are redundant. The expression could be simplified as "self.tickBin >= BINS(self.splits) - 1".

```
48 if (self.tickBin >= ((BINS(self.splits) - 1))) revert InvalidTickBin
      ↪ (self.tickBin);
```

CVF-201 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as: (1 « depth.toUint256()).toInt256() + (stopIdx » (scale - depth).toUint256())

```
110 Quad nextCoef = smallAtDropReader(self, ((1 << depth.toUint256()).
      ↪ toInt256() + (stopIdx >> (scale - depth).toUint256())),
      ↪ deadEra);
```

CVF-202 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Some brackets are redundant. The expression could be simplified as: (POSITIVE_ONE - pool.used.nowAt(pool.era, pool.splits, int256(int24(pool.tickBin))) * pool.deflator) * pool.epsilon / sqrtStrike(pool.splits, pool.tickBin) * pool.fee / exFee(pool.fee) * vars.qStep

```
126 vars.accrual = (
        (
            ((POSITIVE_ONE - pool.used.nowAt(pool.era, pool.splits,
               ↪ int256(int24(pool.tickBin))) * pool.deflator) * pool.
               ↪ epsilon)
                / sqrtStrike(pool.splits, pool.tickBin)
130     ) * pool.fee / exFee(pool.fee)
) * vars.qStep;
```

CVF-203 INFO

- **Category** Procedural
- **Source** Spot.sol

Recommendation Some brackets are redundant. The expression could be simplified as: (POSITIVE_ONE - pool.used.nowAt(pool.era, pool.splits, int256(int24(pool.tickBin))) * pool.deflator) * pool.epsilon * sqrtStrike(pool.splits, pool.tickBin) * pool.fee / exFee(pool.fee) * vars.qStep

```
177 vars.accrual = (
        (
            ((POSITIVE_ONE - pool.used.nowAt(pool.era, pool.splits,
               ↪ int256(int24(pool.tickBin))) * pool.deflator) * pool.
               ↪ epsilon)
180              * sqrtStrike(pool.splits, pool.tickBin)
        ) * pool.fee / exFee(pool.fee)
    ) * vars.qStep;
```

CVF-204 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as: bin < self.latest.start || bin >= self.latest.stop

```
203 if (((bin < self.latest.start) || (bin >= self.latest.stop))) {
```

CVF-205 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as: coarse - subLeftSumDropReader(self, scale, startIdx, deadEra) + subLeftSumDropReader(self, scale, stopIdx, deadEra)

```
95 return ((coarse - subLeftSumDropReader(self, scale, startIdx,
      ↪ deadEra)) + subLeftSumDropReader(self, scale, stopIdx, deadEra
      ↪ ));
```

CVF-206 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as: coarse - subLeftSumNowReader(self, poolEra, scale, startIdx) + subLeftSumNowReader(self, poolEra, scale, stopIdx)

```
438 return ((coarse - subLeftSumNowReader(self, poolEra, scale, startIdx
    ↪ )) + subLeftSumNowReader(self, poolEra, scale, stopIdx));
```

CVF-207 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Some brackets are redundant. The expression could be simplified as: cursor » MIN_SPLITS <= 1

```
54 if (!(((cursor >> MIN_SPLITS) > 1))) revert InvalidAtArguments();
```

```
93 if (!(((cursor >> MIN_SPLITS) > 1))) revert InvalidAtArguments();
```

```
139 if (!(((cursor >> MIN_SPLITS) > 1))) revert InvalidEnterArguments();
```

CVF-208 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Some brackets are redundant. The expression could be simplified as: cursor » MIN_SPLITS == 0

```
126 if (((cursor >> MIN_SPLITS) == 0)) {
```

CVF-209 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Some brackets are redundant. The expression could be simplified as: cursor » MIN_SPLITS == 1

```
42 if (((cursor >> MIN_SPLITS) == 1)) return POSITIVE_ZERO;
```

```
81 if (((cursor >> MIN_SPLITS) == 1)) return POSITIVE_ZERO;
```

CVF-210 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as: eraDay(pool.era) + 2 - params.i

```
236 int256 deadDay = (eraDay(pool.era) + (2 - params.i));
```

CVF-211 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Recommendation Some brackets are redundant. The expression could be simplified as: initial / POSITIVE_EIGHT * (date * POSITIVE_EIGHT - fromInt256(self.lastBucket))

```
22 self.ring[uint256(floorMod(self.lastBucket, 9))] = ((initial /
   ↪ POSITIVE_EIGHT) * (date * POSITIVE_EIGHT - fromInt256(self.
   ↪ lastBucket)));
```

CVF-212 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Some brackets are redundant. The expression could be simplified as: lowSqrtBin(pool, pool.tickBin) - tubLowSqrt(startTub) + pool.epsilon * sqrtStrike(pool.splits, pool.tickBin) * pool.binFrac

```
256 reserve1 = (
        (lowSqrtBin(pool, pool.tickBin) - tubLowSqrt(startTub)) + ((pool
            ↪ .epsilon * sqrtStrike(pool.splits, pool.tickBin)) * pool.
            ↪ binFrac)
    );
```

CVF-213 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as: offset & -1 « scale.toUint256() != 0

```
67 if (((offset & (-1 << scale.toUint256())) != 0)) revert
       ↪ InvalidNowtReaderApplyArguments();
```

CVF-214 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as: offset & -1 « scale.toUint256() != 0

```
261 if (((offset & (-1 << scale.toUint256())) != 0)) revert
       ↪ InvalidLiftReaderApplyArguments();
```

CVF-215 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Some brackets are redundant. The expression could be simplified as: offset < 0 || offset >= int256(1 « scale.toUint256())

```
30 if (((offset < 0) || (offset >= int256(1 << scale.toUint256()))))
      ↪ revert OffsetOutOfRange();
```

CVF-216 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: poolEra < self.deadlineJumps.deadEra(lastJump.toInt8())

```
301 if ((poolEra < self.deadlineJumps.deadEra(lastJump.toInt8())))
      ↪ lastJump = (lastJump - 1);
```

CVF-217 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: poolEra == self.atEra || self.deadlineJumps.nextJump == JUMPS

```
294 if (poolEra == (self.atEra) || ((self.deadlineJumps.nextJump ==
      ↪ JUMPS))) {
```

CVF-218 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Some brackets are redundant. The expression could be simplified as: POSITIVE_ONE / lowSqrtBin(pool, pool.tickBin) - POSITIVE_ONE / tubLowSqrt(stopTub) - pool.epsilon / sqrtStrike(pool.splits, pool.tickBin) * pool.binFrac

```
252 reserve0 = (
        ((POSITIVE_ONE / lowSqrtBin(pool, pool.tickBin)) - (POSITIVE_ONE
            ↪  / tubLowSqrt(stopTub)))
            - ((pool.epsilon / sqrtStrike(pool.splits, pool.tickBin)) *
                ↪ pool.binFrac)
    );
```

CVF-219 INFO

- **Category** Procedural
- **Source** LP.sol

Recommendation Some brackets are redundant. The expression could be simplified as: POSITIVE_ONE / tubLowSqrt(startTub) - POSITIVE_ONE / tubLowSqrt(stopTub)

```
245 reserve0 = ((POSITIVE_ONE / tubLowSqrt(startTub)) - (POSITIVE_ONE /
        ↪ tubLowSqrt(stopTub)));
```

CVF-220 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as: scale > pool.splits

```
474 if (!((scale <= pool.splits))) revert InvalidCreateArguments();
```

CVF-221 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. the expression could be simplified as: self.accrued + self.deadlineJumps.latest * (nextDeflator - poolDeflator)

```
338 self.accrued = (self.accrued + (self.deadlineJumps.latest * (
    ↪ nextDeflator - poolDeflator)));
```

CVF-222 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: self.accrued + self.deadlineJumps.latest * (self.atDeflator - nextDeflator)

```
331 self.accrued = (self.accrued + (self.deadlineJumps.latest * (self.
    ↪ atDeflator - nextDeflator)));
```

CVF-223 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: self.accrued + self.deadlineJumps.latest * (self.atDeflator - poolDeflator)

```
295 self.accrued = (self.accrued + (self.deadlineJumps.latest * (self.
    ↪ atDeflator - poolDeflator)));
```

CVF-224 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: self.deadlineJumps.touched && poolDeflator < self.atDeflator

```
287 if (self.deadlineJumps.touched && (((poolDeflator) < self.atDeflator
    ↪ ))) {
```

CVF-225 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: self.nextJump < JUMPS && poolEra >= self.deadEra(int8(self.nextJump))

```
187 while (self.nextJump < JUMPS && (poolEra >= (self.deadEra(int8(self.
    ↪ nextJump))))) {
```

CVF-226 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as: startIdx < 0 || startIdx > stopIdx || stopIdx > length

```
167 if ((((startIdx < 0) || (startIdx > stopIdx)) || (stopIdx > length))
    ↪ ) revert InvalidAddRangeArguments();
```

CVF-227 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. the expression could be simplified as: startTub >= stopTub

```
100 if (!((startTub < stopTub))) revert InvalidAccruedRangeArguments();
```

CVF-228 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation Some brackets are redundant. The expression could be simplified as: stopIdx & (1 « (scale - depth - 1).toUint256()).toInt256() != 0

```
402 bool branchUp = ((stopIdx & (1 << ((scale - depth) - 1).toUint256())
    ↪ .toInt256()) != 0);
```

CVF-229 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as: stopIdx & int256(1 « (scale - depth - 1).toUint256()) != 0

```
163 bool rightChild = ((stopIdx & int256(1 << ((scale - depth) - 1).
    ↪ toUint256())) != 0);
```

CVF-230 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Recommendation Some brackets are redundant. The expression could be simplified as: stopIdx % (1 « (scale - MIN_SPLITS.toInt256()).toUint256()).toInt256() == 0

```
100 if (((stopIdx % (1 << (scale - MIN_SPLITS.toInt256()).toUint256()).
    ↪ toInt256()) == 0)) {
```

CVF-231 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation Some brackets are redundant. The expression could be simplified as: t1 < JUMPS - 1 ? int8(t1) : JUMPS - 1

```
299 int256 lastJump = (t1 < (JUMPS - 1)) ? int8(t1) : (JUMPS - 1);
```

CVF-232 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Recommendation Some brackets are redundant. The expression could be simplified as: "(int256(1 « scale.toUint256()) + offset « 1) + 1".

```
103 (((int256(1 << scale.toUint256()) + offset) << 1) + 1),
```

```
117 self, pool.era, (((int256(1 << scale.toUint256()) + offset) << 1) +
    ↪ 1), self.ofBelow ? amount : amount.neg(), deadEra, newDeadEra
```

CVF-233 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation Some brackets are redundant. The expression could be simplified as: "offset & -1 « splits.toUint256() != 0" or even as "uint256(offset) » splits.toUint256() != 0".

```
37 if (!(((offset & (-1 << splits.toUint256())) == 0))) revert
   ↪ InvalidAtArguments();
```

```
76 if (!(((offset & (-1 << splits.toUint256())) == 0))) revert
   ↪ InvalidAtArguments();
```

```
115 if (!(((offset & (-1 << splits.toUint256())) == 0))) revert
    ↪ InvalidEnterArguments();
```

CVF-234 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. The expression could be simplified as:"!(scale <= int256(MIN_SPLITS) && 0 <= startIdx && stopIdx <= int256(1 « scale.toUint256()))" or even as "scale > int256(MIN_SPLITS) || 0 > startIdx || stopIdx > int256(1 « scale.toUint256())".

```
65 if (!(((scale <= int256(MIN_SPLITS) && (0 <= startIdx)) && (stopIdx
   ↪ <= int256(1 << scale.toUint256()))))) revert
   ↪ InvalidSumTotalArguments();
```

CVF-235 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Recommendation Some brackets are redundant. This expression could be simplified as: scale > int256(MIN_SPLITS) ? int256(node » (scale - int256(MIN_SPLITS)).toUint256()) : node

```
138 int256 temp = ((scale > int256(MIN_SPLITS)) ? int256(node >> (scale
    ↪ - int256(MIN_SPLITS)).toUint256()) : node);
```

CVF-236 INFO

- **Category** Procedural
- **Source** OptInt256.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Consider specifying as "^0.8.0" or "^0.8.19" if there is something special regarding this particular version. Also relevant for: JumpyFallback.sol, GrowthSplitFrame.sol, BytesLib.sol, DropFaberTotals.sol, EraBoxcarMidSum.sol, JumpyAnchorFaber.sol, UserPay.sol, Utils.sol, SparseFloat.sol, Capper.sol, Payoff.sol, Fees.sol, BoxcarTubFrame.sol, BucketRolling.sol, DailyJumps.sol, DeadlineFlag.sol, EraFaberTotals.sol, DeadlineJumps.sol, GapStagedFrame.sol, EachPayoff.sol, DeadlineHelper.sol, PoolHelper.sol, Spot.sol, LP.sol, Swapper.sol, NewLoan.sol, Structs.sol, Advance.sol, IInfinityPoolPaymentCallback.sol, IInfinityPool.sol, IInfinityPoolDeployer.sol, IInfinityPoolFactory.sol, IInfinityPoolLoanState.sol, IInfinityPoolState.sol, InfinityPoolState.sol, PoolReader.sol, InfinityPoolDeployer.sol, InfinityPoolFactory.sol, InfinityPool.sol, Constants.sol.

```
2 pragma solidity =0.8.19;
```

CVF-237 INFO

- **Category** Suboptimal
- **Source** GrowthSplitFrame.sol

Recommendation The "%" operator could be replaced with a logical "and" to improve efficiency.

```
151 if (((stopIdx % int256(1 << (scale - int256(MIN_SPLITS)).toUint256()
    ↪ )) == 0)) {
```

```
174 if (((stopIdx % int256(1 << (scale - int256(MIN_SPLITS)).toUint256()
    ↪ )) == 0)) {
```

CVF-238 INFO

- **Category** Unclear behavior
- **Source** PoolHelper.sol

Recommendation The "exp" function used here isn't explicitly imported.

```
32 return (logBin(splits) / POSITIVE_TWO).exp();
```

```
40 return (logBin(splits) * (fromInt256(bin) - (fromInt256(BINS(splits)
    ↪ ) / POSITIVE_TWO) + HALF)).exp();
```

```
44 return (logBin(splits) / POSITIVE_TWO * (fromInt256(bin) - (
    ↪ fromInt256(BINS(splits)) / POSITIVE_TWO) + HALF)).exp();
```

```
56 return (logBin(splits) * (fromInt256(tickBin) + binFrac - fromInt256
    ↪ (BINS(splits)) / fromInt256(2))).exp();
```

```
85 return (LOG1PPC * fromInt256(edge)).exp();
```

```
89 return (LOG1PPC / POSITIVE_TWO * fromInt256(edge)).exp();
```

```
93 return (logBin(bin) / POSITIVE_TWO * (fromInt256(bin) - fromInt256(
    ↪ BINS(splits)) / POSITIVE_TWO)).exp();
```

```
114 return ((logTick() / POSITIVE_TWO * fromInt256(tick))).exp();
```

CVF-239 INFO

- **Category** Suboptimal
- **Source** NewLoan.sol

Recommendation The “isInfinity” function could be used to check for infinity.

```
132 // params.lockinEnd != POSITIVE_INFINITY does not work!!
    if ((params.lockinEnd.unwrap() != POSITIVE_INFINITY.unwrap()))
       ↪ revert UnsupportedLockInPeriod(params.lockinEnd);
```

CVF-240 INFO

- **Category** Unclear behavior
- **Source** PoolHelper.sol

Recommendation The “log” function used here isn’t explicitly imported.

```
52 return startPrice.log() / logBin(splits) + fromInt256(BINS(splits))
      ↪ / POSITIVE_TWO;
```

CVF-241 INFO

- **Category** Suboptimal
- **Source** IInfinityPoolFactory.sol

Recommendation The “oldOwner” parameter is redundant, as its value could be derived from the previous event.

```
5 event OwnerChanged(address indexed oldOwner, address indexed
     ↪ newOwner);
```

CVF-242 INFO

- **Category** Suboptimal
- **Source** Constants.sol

Description The "OPT_INT256_NONE" constant is a part of implementation details of the "OptInt256" type and shouldn't by used here.

Recommendation Consider declaring a "none" constant of type "OptInt256" inside the "OptInt256.sol" file, changing the type for the "DEADERA_NONE" constant to "OptInt256" and using the "none" constant declared inside "OptInt256.sol" to initialize "DEADERA_NONE".

```
17 int256 constant DEADERA_NONE = OPT_INT256_NONE;
```

CVF-243 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation The "poolDeflator" argument is redundant, as its value could be derived from the "poolEra" argument.

```
286 function growAccruing(Info storage self, int256 poolEra, Quad
    ↪ poolDeflator) internal {
```

CVF-244 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation The brackets around "lastJump" are redundant.

```
309 while (self.deadlineJumps.nextJump <= (lastJump)) {
```

CVF-245 INFO

- **Category** Unclear behavior
- **Source** DeadlineFlag.sol

Description The code below looks like it is always executed, while actually it is executed only when ”self.touched” is true.

Recommendation Consider putting the rest of the function into an explicit ”else” branch for readability.

```
60 }
```

CVF-246 INFO

- **Category** Unclear behavior
- **Source** PoolReader.sol

Description The code below looks like it is always executed, while actually it is executed only when the stage is not “Join”.

Recommendation Consider putting the rest of the function into an explicit “else” branch.

```
30 }
```

CVF-247 INFO

- **Category** Unclear behavior
- **Source** DeadlineJumps.sol

Description The code below loops like it is always executed while actually it is executed only when ”self.touched” is true.

Recommendation Consider putting the reset of the function into an explicit ”else” branch for readability.

```
56 }
```

CVF-248 INFO

- **Category** Readability
- **Source** Capper.sol

Description The condition looks weird.

Recommendation Consider putting brackets around comparison to make it more readable.

```
31 if (pivotTick <= binLowTick(pool.splits, pool.tickBin) == token) {
```

CVF-249 FIXED

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation The conversion to ”int8” is redundant, as ”self.deadlineJumps.nextJump” is already ”int8”.

Client Comment Simplified as suggested.

```
314 pastJump = int8(self.deadlineJumps.nextJump);
```

CVF-250 INFO

- **Category** Suboptimal
- **Source** BucketRolling.sol

Description The expression “date * POSITIVE_EIGHT” is calculated twice.

Recommendation Consider calculating once and reusing.

```
21 self.lastBucket = floor(date * POSITIVE_EIGHT);
   self.ring[uint256(floorMod(self.lastBucket, 9))] = ((initial /
      ↪ POSITIVE_EIGHT) * (date * POSITIVE_EIGHT - fromInt256(self.
      ↪ lastBucket)));
```

CVF-251 INFO

- **Category** Suboptimal
- **Source** DailyJumps.sol

Description The expression “eraDay(pool.era)” is calculated twice.

Recommendation Consider calculating once and reusing.

```
51 int256 temp = (eraDay(pool.era) - self.atDay);
```

```
57         int256 idx = (eraDay(pool.era) & 1);
```

CVF-252 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Description The expression ”idx.toUint256()” is calculated twice.

Recommendation Consider calculating once and reusing.

```
130 self.drops[idx.toUint256()] = self.drops[idx.toUint256()] + drop;
```

CVF-253 INFO

- **Category** Suboptimal
- **Source** EraBoxcarMidSum.sol

Description The expression ”int256(1 « scale.toUint256())” is calculated twice.

Recommendation Consider calculating once and reusing.

```
30 if (((offset < 0) || (offset >= int256(1 << scale.toUint256()))))
      ↪ revert OffsetOutOfRange();
```

```
32 int256 node = (int256(1 << scale.toUint256()) + offset);
```

CVF-254 INFO

- **Category** Suboptimal
- **Source** BoxcarTubFrame.sol

Description The expression ”int256(1 « scale.toUint256())” is calculated twice.

Recommendation Consider calculating once and reusing.

```
32 if (((offset < 0) || (offset >= int256(1 << scale.toUint256())))
      ↪ revert OffsetOutOfRange();
```

```
34 int256 node = (int256(1 << scale.toUint256()) + offset);
```

CVF-255 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Description The expression ”int256(self.nextJump).toUint256()” is calculated twice.

Recommendation Consider calculating once and reusing.

```
188 self.latest = self.latest - self.drops[(int256(self.nextJump)).
       ↪ toUint256()];
    self.drops[int256(self.nextJump).toUint256()] = POSITIVE_ZERO;
```

CVF-256 INFO

- **Category** Suboptimal
- **Source** BoxcarTubFrame.sol

Description The expression ”node.toUint256” is calculated twice.

Recommendation Consider calculating once and reusing.

```
28 self.coef[node.toUint256()] = self.coef[node.toUint256()] + amount;
```

CVF-257 FIXED

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation The inner brackets are redundant.

Client Comment Simplified as suggested.

```
47 if (!(self.touched)) {
```

CVF-258 FIXED

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation The outer brackets in the assignment are redundant.

Client Comment Simplified as suggested.

```
56 if (jumpyAnchorFaberDataStructure.eraFaberTotals.below) shouldAdd =
   ↪ (bin < validEnd(self, jumpyAnchorFaberDataStructure));
```

```
59 if (shouldAdd) self.halfsum = (self.halfsum + amount);
```

CVF-259 INFO

- **Category** Documentation
- **Source** IInfinityPoolState.sol

Description The particular format of hashed message is unclear.

Recommendation Consider documenting.

```
72 mapping(bytes32 => Capper.Info) capper; // Keyed by (tick, token)
   ↪ hashed
```

CVF-260 INFO

- **Category** Bad datatype
- **Source** IInfinityPool.sol

Recommendation The return type should be “IERC20”.

```
11 function token0() external view returns (address);
   function token1() external view returns (address);
```

CVF-261 INFO

- **Category** Bad datatype
- **Source** InfinityPool.sol

Recommendation The return type should be “IERC20”.

```
274 function token0() public view returns (address) {
```

```
278 function token1() public view returns (address) {
```

CVF-262 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolFactory.sol

Recommendation The return type should be “IInfinityPool”.

```
 9 function getPool(address tokenA, address tokenB, int256 splits)
      ↪ external view returns (address pool);
10 function createPool(address tokenA, address tokenB, int256 splits)
      ↪ external returns (address pool);
```

CVF-263 INFO

- **Category** Bad datatype
- **Source** InfinityPoolDeployer.sol

Recommendation The return type should be “IInfinityPool”.

```
17 function newPool(bytes32 salt) internal virtual returns (address
      ↪ pool) {
```

CVF-264 INFO

- **Category** Bad datatype
- **Source** InfinityPoolFactory.sol

Recommendation The return type should be “IInfinityPool”.

```
28 function createPool(address tokenA, address tokenB, int256 splits)
     ↪ external override returns (address pool) {
```

CVF-265 INFO

- **Category** Documentation
- **Source** InfinityPoolFactory.sol

Description The semantics of the keys is unclear.

Recommendation Consider giving descriptive names to the keys and/or documenting.

```
12 mapping(address => mapping(address => mapping(int256 => address)))
     ↪ public override getPool;
```

CVF-266 INFO

- **Category** Documentation
- **Source** IInfinityPool.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving descriptive names to the returned value and/or documenting.

```
13 function splits() external view returns (int256);
   function newLoan(NewLoan.NewLoanParams memory params, bytes calldata
      ↪  data) external returns (int256, int256);
   function swap(Spot.SwapParams memory params, bytes calldata data)
      ↪ external returns (int256 amount0, int256 amount1);

18     returns (int256 amount0, int256 amount1);

21     returns (int256, int256);
   function getPourQuantities(int256 startTub, int256 stopTub, Quad
      ↪ liquidity) external returns (int256, int256);
   function pour(int256 startTub, int256 stopTub, Quad liquidity, bytes
      ↪  calldata data) external returns (uint256, int256, int256);

26 function drain(int256 lpNum, address receiver, bytes calldata data)
      ↪ external returns (int256, int256);
   function collect(int256 lpNum, address receiver, bytes calldata data
      ↪ ) external returns (int256, int256);
```

CVF-267 INFO

- **Category** Documentation
- **Source** Fees.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
9 function translateFees(Quad f0, Quad f1, PoolState storage pool)
     ↪ internal view returns (int256, int256) {
```

CVF-268 INFO

- **Category** Documentation
- **Source** InfinityPool.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
185 function getPourQuantities(int256 startTub, int256 stopTub, Quad
    ↪ liquidity) external returns (int256, int256) {
```

CVF-269 INFO

- **Category** Documentation
- **Source** SparseFloat.sol

Recommendation The semantics of this function is unclear. It reads from the middle of the sparse significand, so tne output doesn't have much sense in general.

```
162 function read(Info storage self, int128 exponent) internal view
    ↪ returns (FloatBits.Info memory) {
```

CVF-270 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Description The storage slot address of "self.deadlineJumps.drops[int256(self.deadlineJumps.nextJump).toUint256()]" is calculated twice.

Recommendation Consider calculating once and reusing.

```
289 self.deadlineJumps.latest = self.deadlineJumps.latest - self.
    ↪ deadlineJumps.drops[int256(self.deadlineJumps.nextJump).
    ↪ toUint256()];
290 self.deadlineJumps.drops[int256(self.deadlineJumps.nextJump).
    ↪ toUint256()] = POSITIVE_ZERO;
```

CVF-271 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Description The storage slot address of "self.deadlineJumps.drops[int256(self.deadlineJumps.nextJump).toUint256()]" is calculated twice.

Recommendation Consider refactoring to avoid double calculation.

```
310 Quad drop = self.deadlineJumps.drops[int256(self.deadlineJumps.
       ↪ nextJump).toUint256()];
```

```
333     self.deadlineJumps.drops[int256(self.deadlineJumps.nextJump).
           ↪ toUint256()] = POSITIVE_ZERO;
```

CVF-272 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolDeployer.sol

Recommendation The type for the "factory" argument should be "IInfinityPoolFactory".

```
7 function parameters() external view returns (address factory,
     ↪ address token0, address token1, int256 tickSpacing);
```

CVF-273 INFO

- **Category** Bad datatype
- **Source** InfinityPoolDeployer.sol

Recommendation The type for the "factory" argument should be "IInfinityPoolFactory".

```
24 function deploy(address factory, address token0, address token1,
     ↪ int256 splits) internal returns (address pool) {
```

CVF-274 INFO

- **Category** Bad datatype
- **Source** InfinityPoolFactory.sol

Recommendation The type for the “factory” argument should be “IInfinityPoolFactory”.

```
24 function deployPool(address factory, address token0, address token1,
   ↪  int256 splits) internal virtual returns (address pool) {
```

CVF-275 INFO

- **Category** Bad datatype
- **Source** BucketRolling.sol

Recommendation The type for the ”i” variable should be ”uint256”.

```
36 for (int256 i = 0; (i < 9); i = (i + 1)) {
```

CVF-276 INFO

- **Category** Bad datatype
- **Source** BucketRolling.sol

Recommendation The type for the ”idx” variable should be ”uint256”.

```
46 for (int256 idx = 0; (idx < 9); idx = (idx + 1)) {
```

CVF-277 INFO

- **Category** Bad datatype
- **Source** DeadlineFlag.sol

Recommendation The type for the ”oldIndex” variable should be ”uint256”.

```
86 for (int8 oldIndex = self.nextJump; (oldIndex < JUMPS); oldIndex = (
   ↪ oldIndex + 1)) {
```

CVF-278 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolFactory.sol

Recommendation The type for the “pool” parameter should be “IInfinityPool”.

```
6 event PoolCreated(address indexed token0, address indexed token1,
    ↪ int256 splits, address pool);
```

CVF-279 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolFactory.sol

Recommendation The type for the token arguments should be “IERC20”.

```
9  function getPool(address tokenA, address tokenB, int256 splits)
     ↪ external view returns (address pool);
10 function createPool(address tokenA, address tokenB, int256 splits)
     ↪ external returns (address pool);
```

CVF-280 INFO

- **Category** Bad datatype
- **Source** InfinityPoolDeployer.sol

Recommendation The type for the token arguments should be “IERC20”.

```
24 function deploy(address factory, address token0, address token1,
     ↪ int256 splits) internal returns (address pool) {
```

CVF-281 INFO

- **Category** Bad datatype
- **Source** InfinityPoolFactory.sol

Recommendation The type for the token arguments should be “IERC20”.

```
24 function deployPool(address factory, address token0, address token1,
     ↪  int256 splits) internal virtual returns (address pool) {
```

CVF-282 INFO

- **Category** Bad datatype
- **Source** InfinityPoolFactory.sol

Recommendation The type for the token arguments should be “IERC20”.

```
28 function createPool(address tokenA, address tokenB, int256 splits)
      ↪ external override returns (address pool) {
```

CVF-283 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolFactory.sol

Recommendation The type for the token parameters should be “IERC20”.

```
6 event PoolCreated(address indexed token0, address indexed token1,
      ↪ int256 splits, address pool);
```

CVF-284 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolDeployer.sol

Recommendation The type for the tokens argument should be “IERC20”.

```
7 function parameters() external view returns (address factory,
      ↪ address token0, address token1, int256 tickSpacing);
```

CVF-285 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolState.sol

Recommendation The type for these fields should be ”IERC20”.

```
31 address token0;
   address token1;
```

CVF-286 INFO

- **Category** Bad datatype
- **Source** Structs.sol

Recommendation The type for these fields should be “IERC20”.

```
13 address token0;
   address token1;
```

CVF-287 INFO

- **Category** Bad datatype
- **Source** InfinityPoolDeployer.sol

Recommendation The type for these fields should be “IERC20”.

```
10 address token0;
   address token1;
```

CVF-288 INFO

- **Category** Bad datatype
- **Source** IInfinityPoolState.sol

Recommendation The type for this field should be ”IInfinityPoolFactory”.

```
30 address factory;
```

CVF-289 INFO

- **Category** Bad datatype
- **Source** InfinityPoolDeployer.sol

Recommendation The type for this field should be “IInfinityPoolFactory”.

```
9 address factory;
```

CVF-290 INFO

- **Category** Bad datatype
- **Source** InfinityPoolFactory.sol

Recommendation The type of the first two keys should be “IERC20”.

```
12 mapping(address => mapping(address => mapping(int256 => address)))
     ↪ public override getPool;
```

CVF-291 FIXED

- **Category** Bad datatype
- **Source** PoolHelper.sol

Recommendation The value ”1735740000” should be a named constant.

Client Comment Done, using the named constant ”EPOCH”.

```
142 return fromInt256(int256(timestamp) - 1735740000) / fromUint256(1
      ↪ days);
```

CVF-292 INFO

- **Category** Bad datatype
- **Source** DailyJumps.sol

Recommendation The value ”2” should be a named constant.

```
40 if (self.touched && (self.atDay >= ((day - 2)))) {
```

CVF-293 INFO

- **Category** Bad datatype
- **Source** BucketRolling.sol

Recommendation The value ”9” should be a named constant.

```
17 for (uint8 i = 0; i < 9; i++) {
```

```
22 self.ring[uint256(floorMod(self.lastBucket, 9))] = ((initial /
   ↪ POSITIVE_EIGHT) * (date * POSITIVE_EIGHT - fromInt256(self.
   ↪ lastBucket)));
```

```
26 for (uint256 i = 0; i < 9; ++i) {
```

```
34 int256 idx = floorMod((bucket + 1), 9);
```

```
36 for (int256 i = 0; (i < 9); i = (i + 1)) {
```

```
45 if (((bucket - self.lastBucket) >= 9)) {
       for (int256 idx = 0; (idx < 9); idx = (idx + 1)) {
```

```
51         int256 idx = floorMod(passed, 9);
```

```
56     int256 idx = floorMod(bucket, 9);
```

```
61     int256 idx = floorMod(bucket, 9);
```

CVF-294 INFO

- **Category** Bad datatype
- **Source** PoolHelper.sol

Recommendation The value ”fromUint256(1 days)” should be a precomputed named constant.

```
142 return fromInt256(int256(timestamp) - 1735740000) / fromUint256(1
   ↪ days);
```

CVF-295 INFO

- **Category** Suboptimal
- **Source** DeadlineFlag.sol

Description The value “jumpIndex(self, deadEra)” is calculated twice.

Recommendation Consider calculating once and reusing.

```
32 if (jumpIndex(self, deadEra) == JUMP_NONE) return false;
   else return self.drops[int256(jumpIndex(self, deadEra)).toUint256()
      ↪ ];
```

CVF-296 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Description The value “jumpIndex(self, deadEra)” is calculated twice.

Recommendation Consider calculating once and reusing.

```
41 if (jumpIndex(self, deadEra) == JUMP_NONE) return POSITIVE_ZERO;
   else return self.drops[int256(jumpIndex(self, deadEra)).toUint256()
      ↪ ];
```

CVF-297 INFO

- **Category** Suboptimal
- **Source** BucketRolling.sol

Description The value ”pool.date * POSITIVE_EIGHT” is calculated twice.

Recommendation Consider calculating once and reusing.

```
32 int256 bucket = floor((pool.date * POSITIVE_EIGHT));
   Quad fraction = ((pool.date * POSITIVE_EIGHT) - fromInt256(bucket));
```

CVF-298 INFO

- **Category** Suboptimal
- **Source** JumpyFallback.sol

Description The value "self.tree[cursor.toUint256()]" was already calculated at the final loop iteration.

Recommendation Consider reusing the already calculated value.

```
47  DeadlineShaded.Info storage leaf = self.tree[cursor.toUint256()];
60  DeadlineShaded.Info storage trunk = self.tree[cursor.toUint256()];
86  DeadlineShaded.Info storage leaf = self.tree[cursor.toUint256()];
99  DeadlineShaded.Info storage trunk = self.tree[cursor.toUint256()];
134 DeadlineShaded.Info storage trunk = self.tree[cursor.toUint256()];
```

CVF-299 INFO

- **Category** Bad datatype
- **Source** PoolHelper.sol

Recommendation The value 11 should be a named constant.

```
122 return era >> 11;
126 return day << 11;
```

CVF-300 INFO

- **Category** Bad datatype
- **Source** InfinityPoolFactory.sol

Recommendation The value type for this mapping should be "IInfinityPool".

```
12  mapping(address => mapping(address => mapping(int256 => address)))
      ↪ public override getPool;
```

CVF-301 INFO

- **Category** Suboptimal
- **Source** Payoff.sol

Description The values ”liquidity * sqrtStrike” and ”liquidity / sqrtStrike” are calculated in both branches.

Recommendation Consider calculating at one place before the conditional statement.

```
24 pool.flowDot[0][1].create(pool, liquidity * sqrtStrike, scale,
       ↪ offset, deadEra);
   pool.flowDot[0][0].create(pool, liquidity / sqrtStrike, scale,
       ↪ offset, deadEra);
```

```
27 pool.flowDot[1][0].create(pool, liquidity / sqrtStrike, scale,
       ↪ offset, deadEra);
   pool.flowDot[1][1].create(pool, liquidity * sqrtStrike, scale,
       ↪ offset, deadEra);
```

```
39 pool.flowDot[0][1].extend(pool, liquidity * sqrtStrike, scale,
       ↪ offset, oldDeadEra, newDeadEra);
40 pool.flowDot[0][0].extend(pool, liquidity / sqrtStrike, scale,
       ↪ offset, oldDeadEra, newDeadEra);
```

```
42 pool.flowDot[1][0].extend(pool, liquidity / sqrtStrike, scale,
       ↪ offset, oldDeadEra, newDeadEra);
   pool.flowDot[1][1].extend(pool, liquidity * sqrtStrike, scale,
       ↪ offset, oldDeadEra, newDeadEra);
```

CVF-302 FIXED

- **Category** Procedural
- **Source** InfinityPoolState.sol

Description There is no access level specified for this variable, so internal access will be used by default.

Recommendation Consider explicitly specifying an access level.

Client Comment Done, making it internal.

```
9 PoolState pool;
```

CVF-303 INFO

- **Category** Unclear behavior
- **Source** BoxcarTubFrame.sol

Description There is no range check for scale.

Recommendation Consider implementing proper checks.

```
31 function apply_(Info storage self, int256 scale, int256 offset)
   ↪ internal view returns (Quad) {
```

```
54 function addRange(Info storage self, int256 scale, int256 startIdx,
   ↪ int256 stopIdx, Quad change) internal {
```

CVF-304 INFO

- **Category** Unclear behavior
- **Source** DropFaberTotals.sol

Description There is no range check for the ”scale” argument.

Recommendation Consider adding an appropriate check.

```
29 function add(Info storage self, int256 poolEra, int256 scale, int256
   ↪  start, Quad[] memory areas, int256 aDeadEra) internal {
```

CVF-305 INFO

- **Category** Unclear behavior
- **Source** EraFaberTotals.sol

Description There is no range check for the ”scale” argument.

Recommendation Consider adding an appropriate check.

```
50 function sumTotalDropReader(Info storage self, int256 scale, int256
   ↪ startIdx, int256 stopIdx, int256 deadEra) internal view
   ↪ returns (Quad) {
```

```
79 function sumRangeDropReader(Info storage self, int256 scale, int256
   ↪ startIdx, int256 stopIdx, int256 deadEra) internal view
   ↪ returns (Quad) {
```

```
99 function subLeftSumDropReader(Info storage self, int256 scale,
   ↪ int256 stopIdx, int256 deadEra) internal view returns (Quad) {
```

CVF-306 INFO

- **Category** Unclear behavior
- **Source** PoolReader.sol

Description These assignments don’t have any effects, as default values are zero anyway.

```
46 lp.claimedFees0 = 0; // TBD
   lp.claimedFees1 = 0; // TBD
```

CVF-307 FIXED

- **Category** Procedural
- **Source** SparseFloat.sol

Recommendation These commented out lines should be removed.

Client Comment Removed the comment.

```
48 //        int128 exponent = int128(float >> 112);
```

```
55     //            return FloatBits.Info(-16382 - 112, int256(uint256
          ↪ (float & 0x7FFFFFFFFFFFFFFF)));
```

```
61     //            return FloatBits.Info(-16382 - 112, int128( (float
          ↪  & (type(int128).max >> 15)) ));
```

```
70         //                int256(exponent) - 16383 - 112, int256(
              ↪ uint256((float << 15 | 0x8000000000000000) >> 15))
           //                exponent - 16383 - 112, int128(((float <<
              ↪ 15 | type(int128).min) >> 15))
```

CVF-308 FIXED

- **Category** Procedural
- **Source** Capper.sol

Recommendation These commented out lines should be removed.

Client Comment Removed the comment.

```
59 //            SparseFloat.Info storage t1 = EachPayoff.reserveRun[(
      ↪ token == Z ? 0 : 1)];
60 //            Quad t2 = t1.growth(self.entryLevel);
```

CVF-309 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Description These directives are not actually used.

Recommendation Consider removing them.

```
11 using DeadlineJumps for DeadlineSet.Info;
   using DeadlineSet for DeadlineSet.Info;
   using PiecewiseCurve for DeadlineSet.Info;
```

CVF-310 INFO

- **Category** Suboptimal
- **Source** JumpyFallback.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
33 error InvalidAtArguments();
   error InvalidEnterArguments();
```

CVF-311 INFO

- **Category** Suboptimal
- **Source** GrowthSplitFrame.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
28 error InvalidSumTotalArguments();
   error InvalidAccruedRangeArguments();
30 error InvalidAreaArguments();
   error InvalidAddArguments();
   error InvalidSumRangeArguments();
   error InvalidSubLeftSumArguments();
```

CVF-312 INFO

- **Category** Suboptimal
- **Source** EraBoxcarMidSum.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
22 error OffsetOutOfRange();
   error InvalidAddRangeArguments();
```

CVF-313 INFO

- **Category** Suboptimal
- **Source** JumpyAnchorFaber.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
185 error InvalidAddArguments();
    error InvalidSubLeftSumArguments();
    error InvalidCreateArguments();
    error InvalidMoveDropArguments();
    error InvalidSumTotalArguments();
190 error InvalidSumRangeArguments();
```

CVF-314 INFO

- **Category** Suboptimal
- **Source** UserPay.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
18 error UserPayToken0Mismatch();
   error UserPayToken1Mismatch();
```

CVF-315 INFO

- **Category** Suboptimal
- **Source** BoxcarTubFrame.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
16 error OffsetOutOfRange();
   error StartIdOrStopIdOutOfRange();
```

CVF-316 INFO

- **Category** Suboptimal
- **Source** EraFaberTotals.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
23 error InvalidSumTotalArguments();
   error InvalidSumRangeArguments();
   error InvalidSubLeftSumArguments();
```

CVF-317 INFO

- **Category** Suboptimal
- **Source** GapStagedFrame.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
52 error InvalidNowtReaderApplyArguments();
   error InvalidStageArguments();
   error InvalidLiftReaderApplyArguments();
   error InvalidAddRangeArguments();
```

CVF-318 INFO

- **Category** Suboptimal
- **Source** LP.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
65 error InvalidPourArguments();
   error MustWaitTillTailEraToStartDraining(int256 tailEra);
   error LiquidityPositionAlreadyDraining();
   error StageNotJoined();
```

CVF-319 INFO

- **Category** Suboptimal
- **Source** NewLoan.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
46 error InvalidOwedPotential();
```

```
48 error InvalidLendToken();
```

CVF-320 INFO

- **Category** Suboptimal
- **Source** Advance.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
43 error TargetDateInPast();
   error InvalidAlphaOrGamma();
```

CVF-321 INFO

- **Category** Suboptimal
- **Source** InfinityPoolFactory.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
14 error PoolAlreadyExists();
   error InvalidTokens();
   error InvalidSplits();
   error InvalidSender();
```

CVF-322 FIXED

- **Category** Suboptimal
- **Source** JumpyFallback.sol

Description These functions are very similar.

Recommendation Consider refactoring to avoid code duplication.

Client Comment Done, as suggested.

```
36 function nowAt(Info storage self, int256 poolEra, int256 splits,
      ↪ int256 offset) internal returns (Quad) {
```

```
75 function dropAt(Info storage self, int256 splits, int256 offset,
      ↪ int256 deadEra) internal returns (Quad) {
```

CVF-323 FIXED

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation These imports should be merged together.

Client Comment Done, as suggested.

```
14 import {dayEra, eraDay} from "../../libraries/helpers/PoolHelper.sol
      ↪ ";
   import {sqrtStrike} from "../../libraries/helpers/PoolHelper.sol";
```

CVF-324 FIXED

- **Category** Procedural
- **Source** IInfinityPoolState.sol

Recommendation These imports should be merged together.

Client Comment Done, as suggested.

```
6  import {DeadlineJumps} from "../libraries/internal/DeadlineJumps.sol
       ↪ ";

13 import {DropsGroup, DeadlineSet} from "../libraries/internal/
       ↪ DeadlineJumps.sol";
```

CVF-325 FIXED

- **Category** Procedural
- **Source** InfinityPool.sol

Recommendation These imports should be merged together.

Client Comment Done, as suggested.

```
4  import {TWAP_SPREAD_DEFAULT} from "./Constants.sol";
   import {LAMBDA, TWAP_SPREAD_DEFAULT, TUBS} from "./Constants.sol";
```

CVF-326 FIXED

- **Category** Procedural
- **Source** NewLoan.sol

Recommendation These imports should be merged.

Client Comment Done, as suggested.

```
27 import {BINS, binStrike, sqrtStrike, logBin, fluidAt} from "../../
       ↪ libraries/helpers/PoolHelper.sol";

33 import {fluidAt, fracPrice} from "../../libraries/helpers/PoolHelper
       ↪ .sol";
```

CVF-327 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation This assignment does nothing as “eraStep” is guaranteed to be zero already.

```
322 if ((pastJump == 0)) eraStep = 0;
```

CVF-328 FIXED

- **Category** Suboptimal
- **Source** Capper.sol

Description This assignment is identical in both branches.

Recommendation Consider performing it in one place after the conditional statement.

Client Comment Done, as suggested.

```
33 self.lastDrain = pool.era;
```

```
37 self.lastDrain = pool.era;
```

CVF-329 FIXED

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Description This comment looks like an unresolved TODO.

Recommendation Consider either resolving or removing it.

Client Comment Removed the comment.

```
67 //@dev should we make nextJump a uint8?
```

CVF-330 FIXED

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation This commented line should be removed.

Client Comment Removed the comment.

```
150 //            if ((drop != POSITIVE_ZERO)) {
```

CVF-331 FIXED

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation This commented out function should be removed.

Client Comment Removed the comment.

```
90 //    function dropAt(Info storage self, int256 deadEra) internal
   ↪ returns (Anchor.Info storage) {
//        if (jumpIndex(self, deadEra) == JUMP_NONE) {
//            return POSITIVE_ZERO;
//        } else {
//            return self.drops[int256(jumpIndex(self, deadEra)).
   ↪ toUint256()];
//        }
//    }
```

CVF-332 FIXED

- **Category** Procedural
- **Source** Capper.sol

Recommendation This commented out line should be either remove or uncommented.

Client Comment Removed the comment.

```
51 //        self.entryLevel = FloatBits.create(0, SWord.wrap(WORD_ZERO
   ↪ ));
```

CVF-333 FIXED

- **Category** Suboptimal
- **Source** Capper.sol

Recommendation This could be simplified as: return self.entryDeflator != POSITIVE_ZERO || self.lastDrain != 0

Client Comment Simplified as suggested.

```
26 bool notInitialized = self.entryDeflator == POSITIVE_ZERO && self.
      ↪ lastDrain == 0;
   return !notInitialized;
```

CVF-334 INFO

- **Category** Suboptimal
- **Source** DropFaberTotals.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
20 error InvalidAddArguments();
```

CVF-335 INFO

- **Category** Suboptimal
- **Source** JumpyAnchorFaber.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
73 error InvalidDeadline();
```

CVF-336 INFO

- **Category** Suboptimal
- **Source** DailyJumps.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
20 error InvalidDay();
```

CVF-337 INFO

- **Category** Suboptimal
- **Source** DeadlineFlag.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
27 error InvalidDeadline();
```

CVF-338 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
30 error InvalidDeadline();
```

CVF-339 INFO

- **Category** Suboptimal
- **Source** Spot.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
25 error InvalidPushAmount();
```

CVF-340 FIXED

- **Category** Suboptimal
- **Source** Advance.sol

Description This expression appears twice in the code.

Recommendation Consider calculating once before the conditional statement.

Client Comment Optimised as suggested.

```
276 vars.temp = log1p(((params.omega * (pool.binFrac - vars.qSwitch)) /
    ↪ params.flow1start));
```

```
279 vars.t1 = log1p(((params.omega * (pool.binFrac - vars.qSwitch)) /
    ↪ params.flow1start));
```

CVF-341 FIXED

- **Category** Suboptimal
- **Source** Advance.sol

Description This expression appears twice in the code.

Recommendation Consider calculating once before the conditional statement.

Client Comment Optimised as suggested.

```
364 vars.temp = log1p(((params.omega * (pool.binFrac - vars.qSwitch)) /
    ↪ params.flow0start.neg()));
```

```
367 vars.t1 = log1p(((params.omega * (pool.binFrac - vars.qSwitch)) /
    ↪ params.flow0start.neg()));
```

CVF-342 FIXED

- **Category** Suboptimal
- **Source** Advance.sol

Recommendation This expression could be simplified as: vars.freed0 != POSITIVE_ZERO || vars.freed1 != POSITIVE_ZERO

Client Comment Simplified as suggested.

```
115 if (!(((vars.freed0 == POSITIVE_ZERO) && (vars.freed1 ==
    ↪ POSITIVE_ZERO)))) {
```

CVF-343 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Recommendation This field is redundant, as its value could be derived from the “atEra” field.

```
272 Quad atDeflator;
```

CVF-344 INFO

- **Category** Suboptimal
- **Source** DailyJumps.sol

Description This flag seems redundant. It is set to true after the very first update and never reset to false. It probably optimizes some initial operations but makes all the other more expensive. Also it makes the code more complicated.

Recommendation Consider removing this flag.

```
14 bool touched;
```

CVF-345 FIXED

- **Category** Readability
- **Source** DeadlineFlag.sol

Recommendation This is equivalent to: if (drop) {

Client Comment Simplified as suggested.

```
89 if ((drop != false)) {
```

CVF-346 INFO

- **Category** Suboptimal
- **Source** JumpyFallback.sol

Description This library consists only of type declarations.

Recommendation Consider moving the declaration into the top level or turning the library into an interface.

```
10 library DeadlineShaded {
```

CVF-347 INFO

- **Category** Suboptimal
- **Source** DeadlineJumps.sol

Description This library has much in common with the “DeadlineFlag” library.

Recommendation Consider refactoring to reduce code duplication.

```
19 library DeadlineSet {
```

CVF-348 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation This library should be moved into a separate file named ”Anchor.sol”.

```
33 library Anchor {
```

CVF-349 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Recommendation This library should be moved into a separate file named ”AnchorSet.sol”.

```
64 library AnchorSet {
```

CVF-350 INFO

- **Category** Procedural
- **Source** Payoff.sol

Recommendation This library should be moved into a separate file named "AnyPayoff.sol".

```
13 library AnyPayoff {
```

CVF-351 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation This library should be moved into a separate file named "DeadlineSet.sol".

```
19 library DeadlineSet {
```

CVF-352 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Recommendation This library should be moved into a separate file named "DeadlineShaded.sol".

```
10 library DeadlineShaded {
```

CVF-353 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation This library should be moved into a separate file named "DropsGroup.sol".

```
115 library DropsGroup {
```

CVF-354 INFO

- **Category** Procedural
- **Source** SparseFloat.sol

Recommendation This library should be moved into a separate file named "FloatBits.sol".

```
11 library FloatBits {
```

CVF-355 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Recommendation This library should be moved into a separate file named "Gap.sol".

```
17 library Gap {
```

CVF-356 INFO

- **Category** Procedural
- **Source** Swapper.sol

Recommendation This library should be moved into a separate file named "NettingGrowth.sol".

```
20 library NettingGrowth {
```

CVF-357 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation This library should be moved into a separate file named "PiecewiseCurve.sol".

```
176 library PiecewiseCurve {
```

CVF-358 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation This library should be moved into a separate file named "PiecewiseGrowthNew.sol".

```
262 library PiecewiseGrowthNew {
```

CVF-359 INFO

- **Category** Procedural
- **Source** SparseFloat.sol

Recommendation This library should be moved into a separate file named "QuadPacker.sol".

```
43 library QuadPacker {
```

CVF-360 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Recommendation This library should be moved into a separate file named "TailedJumps.sol".

```
243 library TailedJumps {
```

CVF-361 INFO

- **Category** Procedural
- **Source** NewLoan.sol

Recommendation This low-level function should be moved into some utility library.

```
66 function max(Quad x, Quad y) internal pure returns (Quad) {
```

CVF-362 FIXED

- **Category** Suboptimal
- **Source** DeadlineFlag.sol

Recommendation Type conversion is redundant. as ”self.nextJump” is already ”int8”.

Client Comment Removed the redundant conversion as suggested.

```
59 return int8(self.nextJump);
```

CVF-363 FIXED

- **Category** Suboptimal
- **Source** EraBoxcarMidSum.sol

Recommendation Type conversions are redundant as ”pool.tickBin” is already ”int256”.

Client Comment Removed the redundant conversion as suggested.

```
53 return apply_(self, pool.era, pool.splits, ask ? int256(uint256(pool
   ↪ .tickBin + 1)) : int256(uint256(pool.tickBin)));
```

CVF-364 INFO

- **Category** Procedural
- **Source** UserPay.sol

Description We didn’t review these files.

```
4 import {Quad, isPositive, isZero} from "../../types/ABDKMathQuad/
   ↪ Quad.sol";
import {ceil} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-365 INFO

- **Category** Procedural
- **Source** SparseFloat.sol

Description We didn't review these files.

```
4 import {Quad} from "../../types/ABDKMathQuad/ValueType.sol";
  import {POSITIVE_ZERO, NEGATIVE_INFINITY, POSITIVE_INFINITY} from "
    ↪ ../../types/ABDKMathQuad/Constants.sol";
  import {mostSignificantBit} from "../../types/ABDKMathQuad/Helpers.
    ↪ sol";
```

CVF-366 INFO

- **Category** Procedural
- **Source** Fees.sol

Description We didn't review these files.

```
4 import {Quad} from "../../types/ABDKMathQuad/Quad.sol";
  import {floor} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-367 INFO

- **Category** Procedural
- **Source** BucketRolling.sol

Description We didn't review these files.

```
4 import {Quad, POSITIVE_ZERO, POSITIVE_ONE, POSITIVE_EIGHT,
    ↪ fromInt256} from "../../types/ABDKMathQuad/Quad.sol";
  import {floor} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-368 INFO

- **Category** Procedural
- **Source** EachPayoff.sol

Description We didn't review these files.

```
4 import {Quad, POSITIVE_ZERO, POSITIVE_ONE, POSITIVE_TWO, fromInt256,
     ↪  HALF} from "../../types/ABDKMathQuad/Quad.sol";
  import {ceil} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-369 INFO

- **Category** Procedural
- **Source** PoolHelper.sol

Description We didn't review these files.

```
3 import {Quad, fromUint256, fromInt256, intoInt256, POSITIVE_ONE,
     ↪ POSITIVE_TWO, HALF} from "../../types/ABDKMathQuad/Quad.sol";
```

```
11 import {floor} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-370 INFO

- **Category** Procedural
- **Source** Spot.sol

Description We didn't review these files.

```
5 import {Quad, LibOptQuad, POSITIVE_ZERO, POSITIVE_ONE} from "../../
     ↪ types/ABDKMathQuad/Quad.sol";
  import {max, min} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-371 INFO

- **Category** Procedural
- **Source** LP.sol

Description We didn't review these files.

```
4 import {Quad, fromInt256, POSITIVE_ZERO, POSITIVE_ONE, POSITIVE_TWO,
     ↪  HALF} from "../../types/ABDKMathQuad/Quad.sol";
  import {ceil} from "../../types/ABDKMathQuad/Math.sol";
```

CVF-372 INFO

- **Category** Procedural
- **Source** NewLoan.sol

Description We didn't review these files.

```
18 } from "../../types/ABDKMathQuad/Quad.sol";
   import {sqrt, exp} from "../../types/ABDKMathQuad/Math.sol";
20 import "../../types/ABDKMathQuad/MathExtended.sol" as MathExtended;
```

CVF-373 INFO

- **Category** Procedural
- **Source** Advance.sol

Description We didn't review these files.

```
4 import {Quad, fromInt256, intoUint256, POSITIVE_ZERO, POSITIVE_ONE,
     ↪ POSITIVE_TWO, NEGATIVE_TWO} from "../../types/ABDKMathQuad/
     ↪ Quad.sol";
  import {min, max, log, abs, exp2} from "../../types/ABDKMathQuad/
     ↪ Math.sol";
  import {log1p, expm1} from "../../types/ABDKMathQuad/MathExtended.
     ↪ sol";
```

CVF-374 INFO

- **Category** Procedural
- **Source** InfinityPool.sol

Description We didn't review these files.

```
8  import {Quad, POSITIVE_TWO, POSITIVE_FOUR, POSITIVE_EIGHT,
       ↪ fromUint256, fromInt256, LibOptQuad} from "./types/
       ↪ ABDKMathQuad/Quad.sol";

12 import {exp2, floor} from "./types/ABDKMathQuad/Math.sol";
   import {expm1} from "./types/ABDKMathQuad/MathExtended.sol";
```

CVF-375 INFO

- **Category** Procedural
- **Source** JumpyFallback.sol

Description We didn't review this file.

```
4  import {Quad, POSITIVE_ZERO} from "../../types/ABDKMathQuad/Quad.sol
       ↪ ";
```

CVF-376 INFO

- **Category** Procedural
- **Source** GrowthSplitFrame.sol

Description We didn't review this file.

```
4  import {Quad, fromInt256, POSITIVE_ZERO, POSITIVE_TWO} from "../../
       ↪ types/ABDKMathQuad/Quad.sol";
```

CVF-377 INFO

- **Category** Procedural
- **Source** DropFaberTotals.sol

Description We didn't review this file.

```
5  import {Quad, POSITIVE_ZERO} from "../../types/ABDKMathQuad/Quad.sol
       ↪ ";
```

CVF-378 INFO

- **Category** Procedural
- **Source** EraBoxcarMidSum.sol

Description We didn't review this file.

```
4 import {Quad} from "../../types/ABDKMathQuad/Quad.sol";
```

CVF-379 INFO

- **Category** Procedural
- **Source** JumpyAnchorFaber.sol

Description We didn't review this file.

```
4 import {Quad, POSITIVE_ZERO, POSITIVE_TWO} from "../../types/
   ↪ ABDKMathQuad/Quad.sol";
```

CVF-380 INFO

- **Category** Procedural
- **Source** Capper.sol

Description We didn't review this file.

```
4 import {Quad, POSITIVE_ZERO} from "../../types/ABDKMathQuad/Quad.sol
   ↪ ";
```

CVF-381 INFO

- **Category** Procedural
- **Source** Payoff.sol

Description We didn't review this file.

```
5 import {Quad} from "../../types/ABDKMathQuad/Quad.sol";
```

CVF-382 INFO

- **Category** Procedural
- **Source** BoxcarTubFrame.sol

Description We didn't review this file.

```
4 import {Quad} from "../../types/ABDKMathQuad/Quad.sol";
```

CVF-383 INFO

- **Category** Procedural
- **Source** DailyJumps.sol

Description We didn't review this file.

```
4 import {Quad, POSITIVE_ZERO} from "../../types/ABDKMathQuad/Quad.sol
   ↪ ";
```

CVF-384 INFO

- **Category** Procedural
- **Source** EraFaberTotals.sol

Description We didn't review this file.

```
5 import {Quad, POSITIVE_ZERO, POSITIVE_TWO} from "../../types/
   ↪ ABDKMathQuad/Quad.sol";
```

CVF-385 INFO

- **Category** Procedural
- **Source** DeadlineJumps.sol

Description We didn't review this file.

```
4 import {Quad, POSITIVE_ZERO, wrap} from "../../types/ABDKMathQuad/
   ↪ Quad.sol";
```

CVF-386 INFO

- **Category** Procedural
- **Source** GapStagedFrame.sol

Description We didn't review this file.

```
4 import {Quad, POSITIVE_ZERO} from "../../types/ABDKMathQuad/Quad.sol
    ↪ ";
```

CVF-387 INFO

- **Category** Procedural
- **Source** Swapper.sol

Description We didn't review this file.

```
4 import {Quad, fromInt256, POSITIVE_ZERO, POSITIVE_ONE, fromUint256}
    ↪ from "../../types/ABDKMathQuad/Quad.sol";
```

CVF-388 INFO

- **Category** Procedural
- **Source** IInfinityPool.sol

Description We didn't review this file.

```
4 import {Quad} from "../types/ABDKMathQuad/Quad.sol";
```

CVF-389 INFO

- **Category** Procedural
- **Source** IInfinityPoolDeployer.sol

Description We didn't review this file.

```
4 import {Quad} from "../types/ABDKMathQuad/Quad.sol";
```

CVF-390 INFO

- **Category** Procedural
- **Source** IInfinityPoolLoanState.sol

Description We didn't review this file.

```
4 import {Quad} from "../types/ABDKMathQuad/Quad.sol";
```

CVF-391 INFO

- **Category** Procedural
- **Source** IInfinityPoolState.sol

Description We didn't review this file.

```
4 import {Quad, POSITIVE_ZERO} from "../types/ABDKMathQuad/Quad.sol";
```

CVF-392 INFO

- **Category** Procedural
- **Source** InfinityPoolState.sol

Description We didn't review this file.

```
4 import {Quad, fromUint256, fromInt256} from "./types/ABDKMathQuad/
    ↪ Quad.sol";
```

CVF-393 INFO

- **Category** Procedural
- **Source** PoolReader.sol

Description We didn't review this file.

```
7 import {Quad} from "./types/ABDKMathQuad/Quad.sol";
```

CVF-394 INFO

- **Category** Procedural
- **Source** InfinityPoolDeployer.sol

Description We didn't review this file.

```
5 import {Quad} from "./types/ABDKMathQuad/Quad.sol";
```

CVF-395 INFO

- **Category** Procedural
- **Source** Constants.sol

Description We didn't review this file.

```
4 import {Quad} from "./types/ABDKMathQuad/Quad.sol";
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting