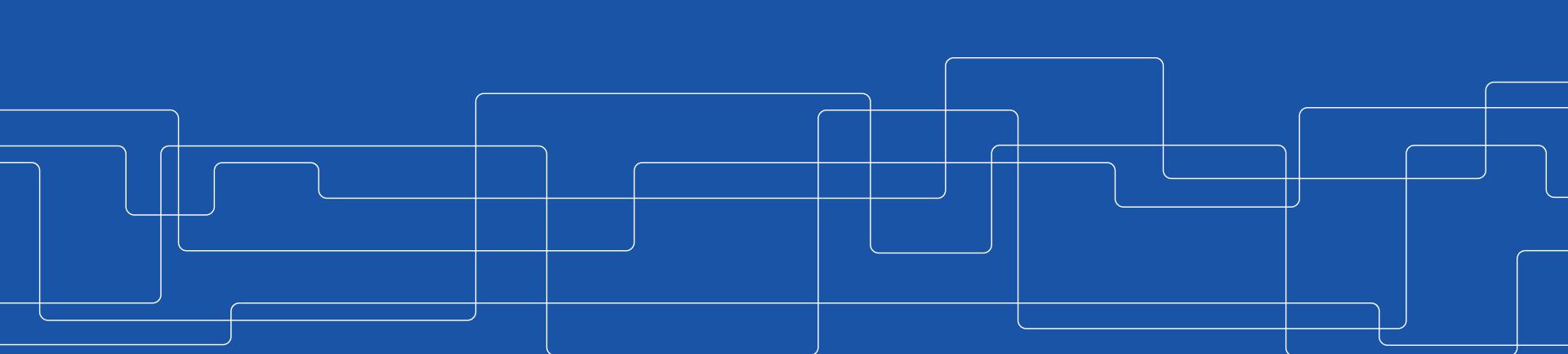




Augmenting Diffs With Runtime Information

Khashayar Etemadi, Aman Sharma, Fernanda Madeiral, Martin Monperrus
April, 2024





Outline

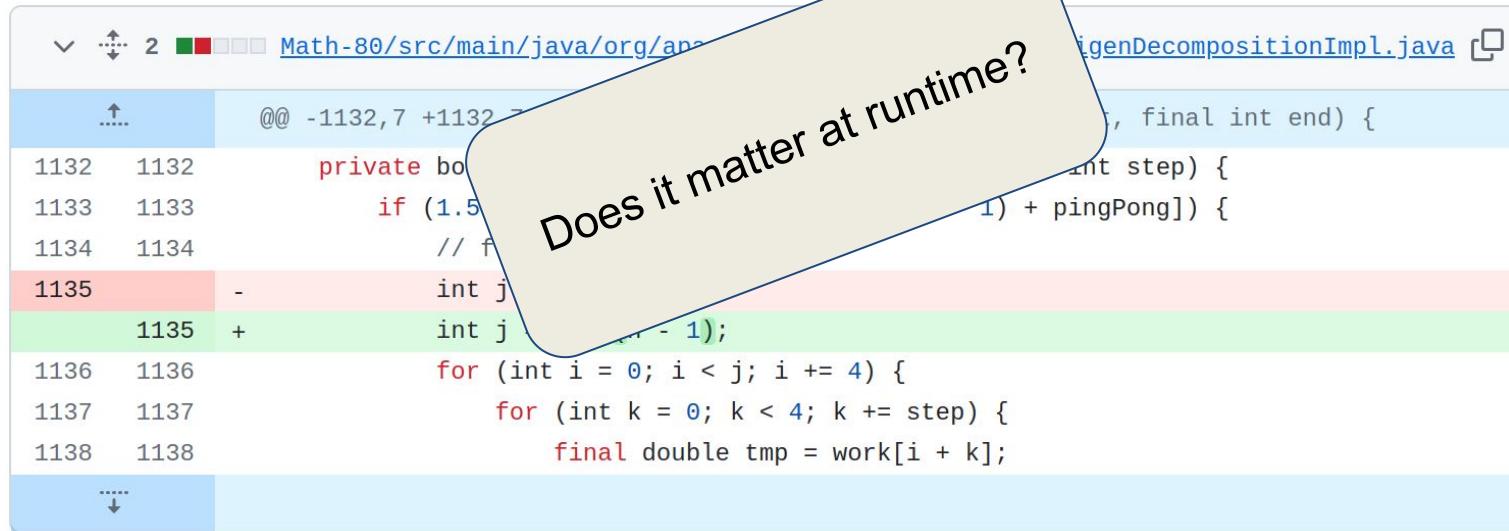
- Code Diff Incompleteness
- Improving Code Diff
- Augmentation by Collector-Sahab
- Technical Details
- Experiments
- Takeaway and Future Work



Code Diff Incompleteness

```
@@ -1132,7 +1132,7 @@ private int goodStep(final int start, final int end) {  
1132 1132     private boolean flipIfWarranted(final int n, final int step) {  
1133 1133         if (1.5 * work[pingPong] < work[4 * (n - 1) + pingPong]) {  
1134 1134             // flip array  
1135 -         int j = 4 * n - 1;  
1135 +         int j = 4 * (n - 1);  
1136 1136             for (int i = 0; i < j; i += 4) {  
1137 1137                 for (int k = 0; k < 4; k += step) {  
1138 1138                     final double tmp = work[i + k];  
.....  
↓
```

Code Diff Incompleteness



Does it matter at runtime?

Math-80/src/main/java/org/apache/commons/math3/optimization/neldermead/AbstractSimplex.java

```
@@ -1132,7 +1132,7 @@ private boolean isSimplexConverged(Simplex simplex, final int end) {  
     int step) {  
         int[] pingPong = new int[simplex.getDim()];  
         for (int i = 0; i < simplex.getDim(); i++) {  
             pingPong[i] = (i + pingPong[i]) % (simplex.getDim() - 1);  
         }  
         for (int i = 0; i < j; i += 4) {  
             for (int k = 0; k < 4; k += step) {  
                 final double tmp = work[i + k];  
                 work[i + k] = work[pingPong[i + k]];  
                 work[pingPong[i + k]] = tmp;  
             }  
         }  
     }  
 }
```

Improving Code Diff

➤ Mergely [1]

- Extracts changes at code element level

```
1 private boolean flipIfWarranted(final int n, final int step) {  
2     if (1.5 * work[pingPong] < work[4 * (n - 1) + pingPong]) {  
3         // flip array  
4         int j = 4 * n - 1;  
5         for (int i = 0; i < j; i += 4) {  
6             for (int k = 0; k < 4; k += step) {  
7                 final double tmp = work[i + k];  
8                 work[i + k] = work[j - k];  
9                 work[j - k] = tmp;  
10            }  
11            j -= 4;  
12        }  
13        return true;  
14    }  
15    return false;  
16 }
```

```
1 private boolean flipIfWarranted(final int n, final int step) {  
2     if (1.5 * work[pingPong] < work[4 * (n - 1) + pingPong]) {  
3         // flip array  
4         int j = 4 * (n - 1);  
5         for (int i = 0; i < j; i += 4) {  
6             for (int k = 0; k < 4; k += step) {  
7                 final double tmp = work[i + k];  
8                 work[i + k] = work[j - k];  
9                 work[j - k] = tmp;  
10            }  
11            j -= 4;  
12        }  
13        return true;  
14    }  
15    return false;  
16 }
```



Improving Code Diff

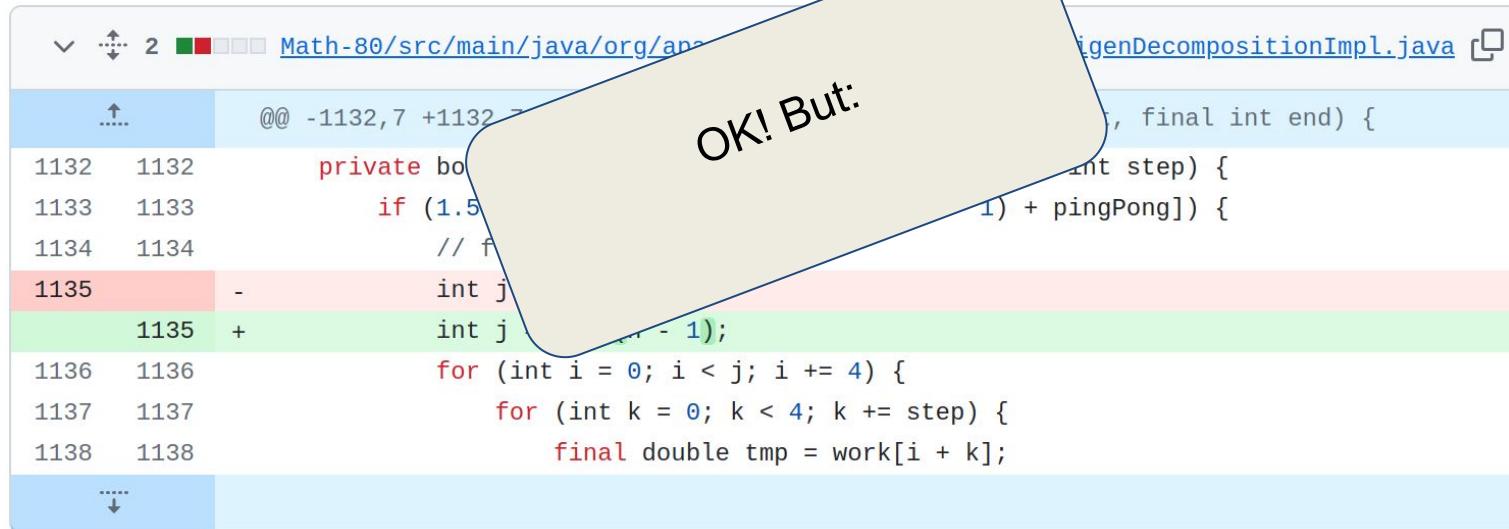
- Mergely [1]
 - Extracts changes at code element level
- GumTree [2]
 - Computes minimum ast modifications



Improving Code Diff

- Mergely [1]
 - Extracts changes at code element level
- GumTree [2]
 - Computes minimum ast modifications
- srcDIFF [3]
 - Produces diff similar to developer's changes

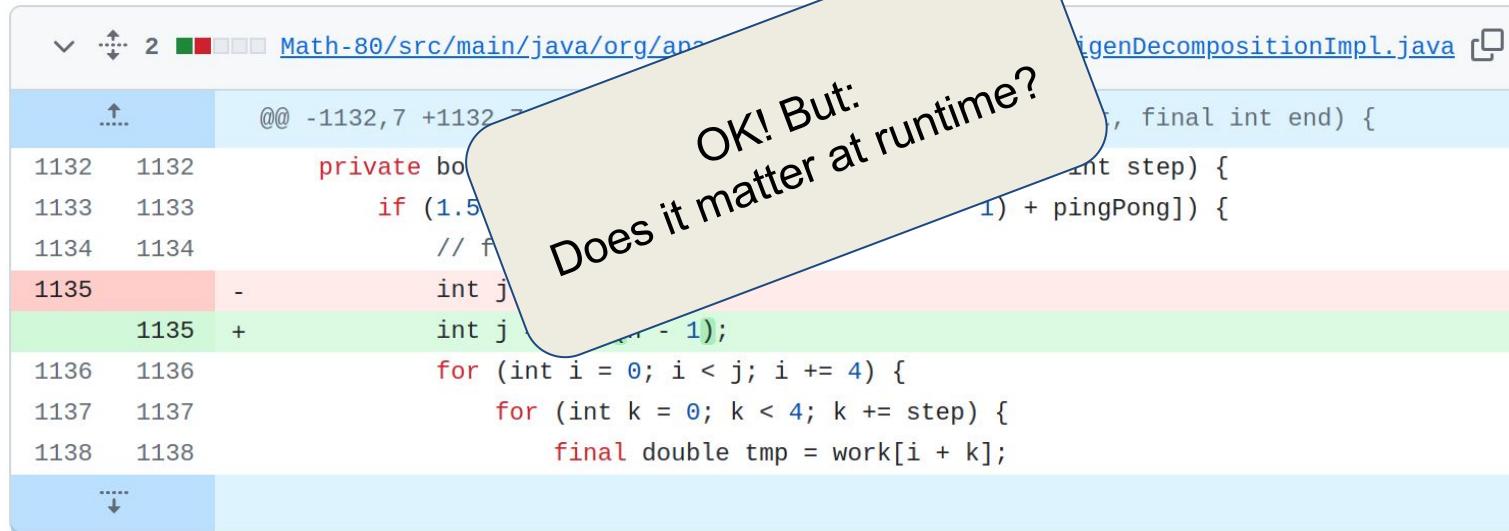
Augmentation by Collector-Sahab



OK! But:

```
@@ -1132,7 +1132,7
 1132 1132     private boolean pingPong;
 1133 1133         if (1.5 < step) {
 1134 1134             // f
 1135 1135 -         int j = i + step - 1;
 1135 1135 +         int j = i + step - 1);
 1136 1136             for (int i = 0; i < j; i += 4) {
 1137 1137                 for (int k = 0; k < 4; k += step) {
 1138 1138                     final double tmp = work[i + k];
```

Augmentation by Collector-Sahab



OK! But:
Does it matter at runtime?

```
@@ -1132,7 +1132,7 @@  
 1132 1132     private boolean pingPong;  
 1133 1133         if (1.5 *  
 1134 1134             // f  
 1135 1135 -         int j = i + step - 1;  
 1135 1135 +         int j = i + step - 1);  
 1136 1136             for (int i = 0; i < j; i += 4) {  
 1137 1137                 for (int k = 0; k < 4; k += step) {  
 1138 1138                     final double tmp = work[i + k];  
 1139 1139                     if (tmp >= 0.0) {  
 1140 1140                         pingPong = !pingPong;  
 1141 1141                         if (tmp >= 1.0) {  
 1142 1142                             pingPong = !pingPong;  
 1143 1143                         if (tmp >= 1.0) {  
 1144 1144                             pingPong = !pingPong;  
 1145 1145                         if (tmp >= 1.0) {  
 1146 1146                             pingPong = !pingPong;  
 1147 1147                         if (tmp >= 1.0) {  
 1148 1148                             pingPong = !pingPong;  
 1149 1149                         if (tmp >= 1.0) {  
 1150 1150                             pingPong = !pingPong;  
 1151 1151                         if (tmp >= 1.0) {  
 1152 1152                             pingPong = !pingPong;  
 1153 1153                         if (tmp >= 1.0) {  
 1154 1154                             pingPong = !pingPong;  
 1155 1155                         if (tmp >= 1.0) {  
 1156 1156                             pingPong = !pingPong;  
 1157 1157                         if (tmp >= 1.0) {  
 1158 1158                             pingPong = !pingPong;  
 1159 1159                         if (tmp >= 1.0) {  
 1160 1160                             pingPong = !pingPong;  
 1161 1161                         if (tmp >= 1.0) {  
 1162 1162                             pingPong = !pingPong;  
 1163 1163                         if (tmp >= 1.0) {  
 1164 1164                             pingPong = !pingPong;  
 1165 1165                         if (tmp >= 1.0) {  
 1166 1166                             pingPong = !pingPong;  
 1167 1167                         if (tmp >= 1.0) {  
 1168 1168                             pingPong = !pingPong;  
 1169 1169                         if (tmp >= 1.0) {  
 1170 1170                             pingPong = !pingPong;  
 1171 1171                         if (tmp >= 1.0) {  
 1172 1172                             pingPong = !pingPong;  
 1173 1173                         if (tmp >= 1.0) {  
 1174 1174                             pingPong = !pingPong;  
 1175 1175                         if (tmp >= 1.0) {  
 1176 1176                             pingPong = !pingPong;  
 1177 1177                         if (tmp >= 1.0) {  
 1178 1178                             pingPong = !pingPong;  
 1179 1179                         if (tmp >= 1.0) {  
 1180 1180                             pingPong = !pingPong;  
 1181 1181                         if (tmp >= 1.0) {  
 1182 1182                             pingPong = !pingPong;  
 1183 1183                         if (tmp >= 1.0) {  
 1184 1184                             pingPong = !pingPong;  
 1185 1185                         if (tmp >= 1.0) {  
 1186 1186                             pingPong = !pingPong;  
 1187 1187                         if (tmp >= 1.0) {  
 1188 1188                             pingPong = !pingPong;  
 1189 1189                         if (tmp >= 1.0) {  
 1190 1190                             pingPong = !pingPong;  
 1191 1191                         if (tmp >= 1.0) {  
 1192 1192                             pingPong = !pingPong;  
 1193 1193                         if (tmp >= 1.0) {  
 1194 1194                             pingPong = !pingPong;  
 1195 1195                         if (tmp >= 1.0) {  
 1196 1196                             pingPong = !pingPong;  
 1197 1197                         if (tmp >= 1.0) {  
 1198 1198                             pingPong = !pingPong;  
 1199 1199                         if (tmp >= 1.0) {  
 1200 1200                             pingPong = !pingPong;  
 1201 1201                         if (tmp >= 1.0) {  
 1202 1202                             pingPong = !pingPong;  
 1203 1203                         if (tmp >= 1.0) {  
 1204 1204                             pingPong = !pingPong;  
 1205 1205                         if (tmp >= 1.0) {  
 1206 1206                             pingPong = !pingPong;  
 1207 1207                         if (tmp >= 1.0) {  
 1208 1208                             pingPong = !pingPong;  
 1209 1209                         if (tmp >= 1.0) {  
 1210 1210                             pingPong = !pingPong;  
 1211 1211                         if (tmp >= 1.0) {  
 1212 1212                             pingPong = !pingPong;  
 1213 1213                         if (tmp >= 1.0) {  
 1214 1214                             pingPong = !pingPong;  
 1215 1215                         if (tmp >= 1.0) {  
 1216 1216                             pingPong = !pingPong;  
 1217 1217                         if (tmp >= 1.0) {  
 1218 1218                             pingPong = !pingPong;  
 1219 1219                         if (tmp >= 1.0) {  
 1220 1220                             pingPong = !pingPong;  
 1221 1221                         if (tmp >= 1.0) {  
 1222 1222                             pingPong = !pingPong;  
 1223 1223                         if (tmp >= 1.0) {  
 1224 1224                             pingPong = !pingPong;  
 1225 1225                         if (tmp >= 1.0) {  
 1226 1226                             pingPong = !pingPong;  
 1227 1227                         if (tmp >= 1.0) {  
 1228 1228                             pingPong = !pingPong;  
 1229 1229                         if (tmp >= 1.0) {  
 1230 1230                             pingPong = !pingPong;  
 1231 1231                         if (tmp >= 1.0) {  
 1232 1232                             pingPong = !pingPong;  
 1233 1233                         if (tmp >= 1.0) {  
 1234 1234                             pingPong = !pingPong;  
 1235 1235                         if (tmp >= 1.0) {  
 1236 1236                             pingPong = !pingPong;  
 1237 1237                         if (tmp >= 1.0) {  
 1238 1238                             pingPong = !pingPong;  
 1239 1239                         if (tmp >= 1.0) {  
 1240 1240                             pingPong = !pingPong;  
 1241 1241                         if (tmp >= 1.0) {  
 1242 1242                             pingPong = !pingPong;  
 1243 1243                         if (tmp >= 1.0) {  
 1244 1244                             pingPong = !pingPong;  
 1245 1245                         if (tmp >= 1.0) {  
 1246 1246                             pingPong = !pingPong;  
 1247 1247                         if (tmp >= 1.0) {  
 1248 1248                             pingPong = !pingPong;  
 1249 1249                         if (tmp >= 1.0) {  
 1250 1250                             pingPong = !pingPong;  
 1251 1251                         if (tmp >= 1.0) {  
 1252 1252                             pingPong = !pingPong;  
 1253 1253                         if (tmp >= 1.0) {  
 1254 1254                             pingPong = !pingPong;  
 1255 1255                         if (tmp >= 1.0) {  
 1256 1256                             pingPong = !pingPong;  
 1257 1257                         if (tmp >= 1.0) {  
 1258 1258                             pingPong = !pingPong;  
 1259 1259                         if (tmp >= 1.0) {  
 1260 1260                             pingPong = !pingPong;  
 1261 1261                         if (tmp >= 1.0) {  
 1262 1262                             pingPong = !pingPong;  
 1263 1263                         if (tmp >= 1.0) {  
 1264 1264                             pingPong = !pingPong;  
 1265 1265                         if (tmp >= 1.0) {  
 1266 1266                             pingPong = !pingPong;  
 1267 1267                         if (tmp >= 1.0) {  
 1268 1268                             pingPong = !pingPong;  
 1269 1269                         if (tmp >= 1.0) {  
 1270 1270                             pingPong = !pingPong;  
 1271 1271                         if (tmp >= 1.0) {  
 1272 1272                             pingPong = !pingPong;  
 1273 1273                         if (tmp >= 1.0) {  
 1274 1274                             pingPong = !pingPong;  
 1275 1275                         if (tmp >= 1.0) {  
 1276 1276                             pingPong = !pingPong;  
 1277 1277                         if (tmp >= 1.0) {  
 1278 1278                             pingPong = !pingPong;  
 1279 1279                         if (tmp >= 1.0) {  
 1280 1280                             pingPong = !pingPong;  
 1281 1281                         if (tmp >= 1.0) {  
 1282 1282                             pingPong = !pingPong;  
 1283 1283                         if (tmp >= 1.0) {  
 1284 1284                             pingPong = !pingPong;  
 1285 1285                         if (tmp >= 1.0) {  
 1286 1286                             pingPong = !pingPong;  
 1287 1287                         if (tmp >= 1.0) {  
 1288 1288                             pingPong = !pingPong;  
 1289 1289                         if (tmp >= 1.0) {  
 1290 1290                             pingPong = !pingPong;  
 1291 1291                         if (tmp >= 1.0) {  
 1292 1292                             pingPong = !pingPong;  
 1293 1293                         if (tmp >= 1.0) {  
 1294 1294                             pingPong = !pingPong;  
 1295 1295                         if (tmp >= 1.0) {  
 1296 1296                             pingPong = !pingPong;  
 1297 1297                         if (tmp >= 1.0) {  
 1298 1298                             pingPong = !pingPong;  
 1299 1299                         if (tmp >= 1.0) {  
 1300 1300                             pingPong = !pingPong;  
 1301 1301                         if (tmp >= 1.0) {  
 1302 1302                             pingPong = !pingPong;  
 1303 1303                         if (tmp >= 1.0) {  
 1304 1304                             pingPong = !pingPong;  
 1305 1305                         if (tmp >= 1.0) {  
 1306 1306                             pingPong = !pingPong;  
 1307 1307                         if (tmp >= 1.0) {  
 1308 1308                             pingPong = !pingPong;  
 1309 1309                         if (tmp >= 1.0) {  
 1310 1310                             pingPong = !pingPong;  
 1311 1311                         if (tmp >= 1.0) {  
 1312 1312                             pingPong = !pingPong;  
 1313 1313                         if (tmp >= 1.0) {  
 1314 1314                             pingPong = !pingPong;  
 1315 1315                         if (tmp >= 1.0) {  
 1316 1316                             pingPong = !pingPong;  
 1317 1317                         if (tmp >= 1.0) {  
 1318 1318                             pingPong = !pingPong;  
 1319 1319                         if (tmp >= 1.0) {  
 1320 1320                             pingPong = !pingPong;  
 1321 1321                         if (tmp >= 1.0) {  
 1322 1322                             pingPong = !pingPong;  
 1323 1323                         if (tmp >= 1.0) {  
 1324 1324                             pingPong = !pingPong;  
 1325 1325                         if (tmp >= 1.0) {  
 1326 1326                             pingPong = !pingPong;  
 1327 1327                         if (tmp >= 1.0) {  
 1328 1328                             pingPong = !pingPong;  
 1329 1329                         if (tmp >= 1.0) {  
 1330 1330                             pingPong = !pingPong;  
 1331 1331                         if (tmp >= 1.0) {  
 1332 1332                             pingPong = !pingPong;  
 1333 1333                         if (tmp >= 1.0) {  
 1334 1334                             pingPong = !pingPong;  
 1335 1335                         if (tmp >= 1.0) {  
 1336 1336                             pingPong = !pingPong;  
 1337 1337                         if (tmp >= 1.0) {  
 1338 1338                             pingPong = !pingPong;
```

Augmentation by Collector-Sahab

```
1132 1132      private boolean flipIfWarranted(final int n, final int step) {  
1133 1133          if (1.5 * work[pingPong] < work[4 * (n - 1) + pingPong]) {  
1134 1134              // flip array  
1135 -          int j = 4 * n - 1;  
1135 +          int j = 4 * (n - 1);  
1136 1136          for (int i = 0; i < j; i += 4) {
```



COLLECTOR-SAHAB / differentiating test: [EigenDecompositionImplTest](#)

j=27 only occurs in the original version.



COLLECTOR-SAHAB / differentiating test: [EigenDecompositionImplTest](#)

j=24 only occurs in the patched version.

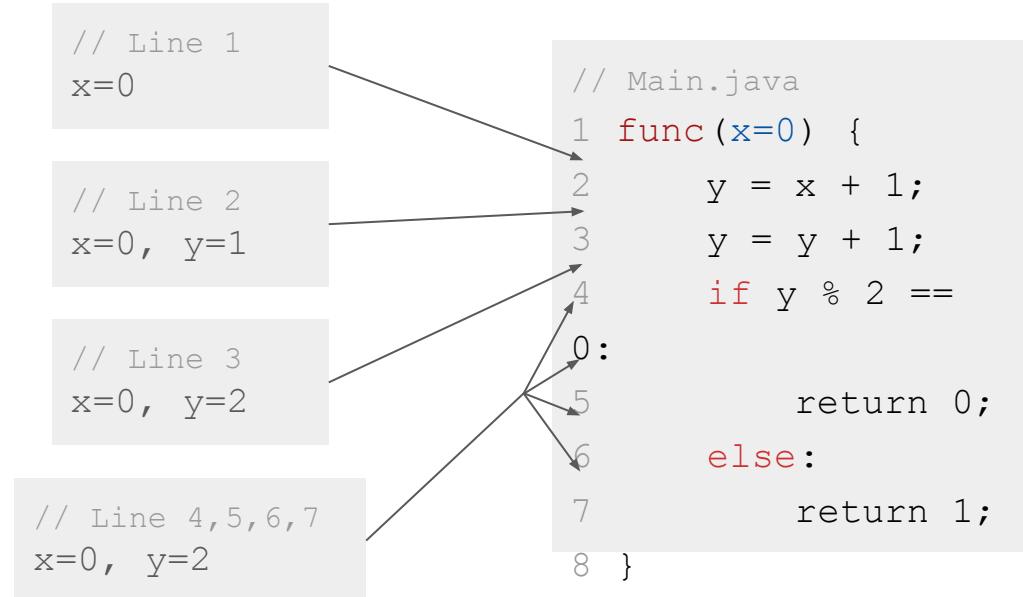
```
1137 1137          for (int k = 0; k < 4; k += step) {  
1138 1138              final double tmp = work[i + k];  
1139 1139              work[i + k] = work[j - k];  
1140 1140              work[j - k] = tmp;  
1141 1141          }  
1142 1142          j -= 4;
```

Augmentation by Collector-Sahab

```
1132 1132      private boolean flipIfWarranted(boolean flip) {  
1133 1133          if (1.5 * work[pingPong] < 0) {  
1134 1134              // flip array  
1135 -                  int i = pingPong + 1;  
1135 +                  int i = pingPong - 1;  
1136 1136          }  
  
COLLECTOR: j=27 only occurs in the original version.  
COLLECTOR: j=24 differentiating test: EigenDecompositionImplTest  
j=24 only occurs in the patched version.  
  
1137 1137          for (int k = 0; k < 4; k += step) {  
1138 1138              final double tmp = work[i + k];  
1139 1139              work[i + k] = work[j - k];  
1140 1140              work[j - k] = tmp;  
1141 1141          }  
1142 1142          j -= 4;
```

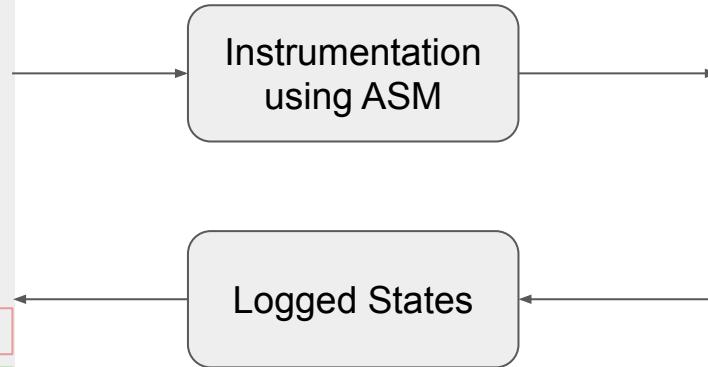
Technical Concepts: Program States

Program states are values of visible variables at each line.



Technical Concepts: Workflow

```
// Main.java
1 func (x=0) {
2     y = x + 1;
3 -     y = y + 1;
3 +     y = y + 2;
4     if y % 2 ==
y = 2 only before
y = 3 only after
5
6         return 0;
7     else:
8         return 1;
```



<pre>// Line 2 x=0, y=1 // No diff</pre>	<pre>// Line 4 x=0, y=2 x=0, y=3</pre>
--	--

```
// Main.java
1 func (x=0) {
2     y = x + 1;
logState()
3 -     y = y + 1;
3 +     y = y + 2;
4     if y % 2 ==
logState()
logState()
6     else:
logState()
8 }
```



Experiments Results

Dataset: 587

Plausible APR Patches
for Defects4J



Experiments Results

Dataset: 587

Plausible APR Patches
for Defects4J

95% (555/587)

Augmented by Collector-Sahab



Experiments Results

Dataset: 587

Plausible APR Patches
for Defects4J

95% (555/587)

Augmented by Collector-Sahab

Users find Augmentations
Useful | Clear | Novel



Takeaways & Future Work

Collector-Sahab effectively detects fine-grained runtime diffs and provides useful augmentations



Takeaway & Future Work

Collector-Sahab effectively detects fine-grained runtime diffs and provides useful augmentations

How can we detect and discard spurious runtime differences?

References

1. Mergely. (2022) Diff text documents online with mergely, an editor and html5 javascript library. [Online]. Available: <https://www.mergely.com/>
2. Falleri, J. R., Morandat, F., Blanc, X., Martinez, M., & Monperrus, M. (2014, September). Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering* (pp. 313-324).
3. Decker, M. J., Collard, M. L., Volkert, L. G., & Maletic, J. I. (2020). srcDiff: A syntactic differencing approach to improve the understandability of deltas. *Journal of Software: Evolution and Process*, 32(4), e2226.

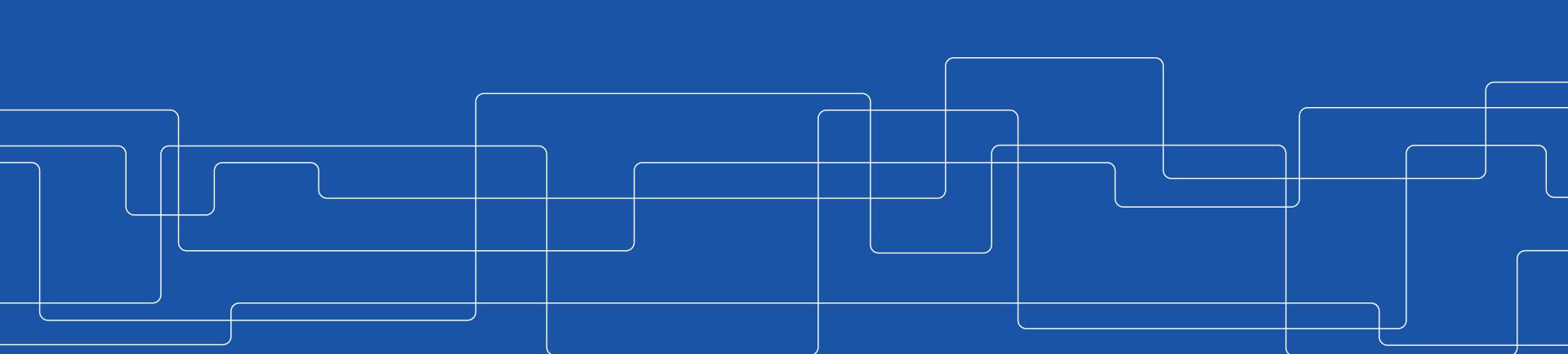


Thanks for listening!

Khashayar Etemadi

khaes@kth.se

<https://github.com/ASSERT-KTH/collector-sahab>





Q&A

Etemadi, K., Sharma, A., Madeiral, F., & Monperrus, M. (2023). Augmenting Diffs With Runtime Information. *IEEE Transactions on Software Engineering*.

<https://github.com/ASSERT-KTH/collector-sahab>