Mining Oracles for Fully Automated Test Generation

Andreas Zeller
Saarland University, Saarbrücken, Germany
Testing
Software is manifold
Software is manifold
Software is manifold
Software is manifold
Software is manifold
Software is manifold
Testing

Configurations
Testing

Configurations
Dijkstra’s Curse

Testing can only find the presence of errors, not their absence
Formal Verification
Formal Verification

Abstraction

Configurations
Formal Verification

Abstraction

Configurations
The Best of two Worlds
<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>E-mail</th>
<th>Phone</th>
<th>Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>James S.</td>
<td>Roebuck</td>
<td>JamesSRoe@ex</td>
<td>561-888-1230</td>
<td>561-888-1230</td>
</tr>
<tr>
<td>Naomi D.</td>
<td>Long</td>
<td>NaomiDLo@ex</td>
<td>390-12-567</td>
<td>390-12-1234</td>
</tr>
<tr>
<td>Karen L.</td>
<td>Lloyd</td>
<td>KarenLLloyd@ex</td>
<td>228-76-1230</td>
<td>228-76-8710</td>
</tr>
<tr>
<td>Jean R.</td>
<td>Voigt</td>
<td>JeanRVoigt@ex</td>
<td>610-344-1230</td>
<td>610-344-1230</td>
</tr>
<tr>
<td>Douglas L.</td>
<td>Green</td>
<td>DouglasLG@ex</td>
<td>612-615-1230</td>
<td>612-615-8710</td>
</tr>
</tbody>
</table>

**Contractors**
- Europe
- U.S.

**New contact**
- First name: Karen L.
- Last name: Lloyd
- E-Mail: KarenLLloyd@ex
- Phone: 228-76-1230
- Mobile: 228-76-8710
- Notes: 1673 Jehovah Drive, Fredericksburg, VA 22408
Infinite Monkey Theorem
Evolutionary Algorithms

- Create population
- Create mutations
- Rank
- Select
- Recombine
Create population

```
“fdsakfh+ew%3gfhd%4f”
```

```
“fwe8^ru786234jä”
```

Mutation

```
“fdsakfh+br%3gfhd%4f”
```

```
“fwe8^ru&26234jä”
```

```
“fdsakfh+ew%4gfhd%4f”
```

```
“xb3#ru786234jä”
```

Recombine

```
“fdsakfh+ew%4gfhd%4f”
```

```
“xb3#ru786234jä”
```
Create population

```
“fdsakfh+ew%3gfhd4i%4f”
```

```
“fwe8^ru786234jä”
```

Mutation

```
“fdsakfh+br%3gfhd4i%4f”
```

```
“fdsakfh+ew%4gfhd4i%4f”
```

```
“fwe8^ru&26234jä”
```

```
“xb3#ru786234jä”
```

Crossover

```
“fdsakfh+ew%4gfhd4i%4f”
```

```
“xb3#ru786234jä”
```

```
“xb3#ru7%4gfhd4i%4f”
```
Selection and Ranking

if (angle = 47 \land power = 532) \{ \ldots \}
Selection and Ranking

if (angle = 47 ∧ power = 532) { ... }

angle = 51
angle = 48
angle = 47
EXSYST

Florian Groß, Andreas Želler
Today's Keynote: Search-Based Program Analysis
Test Coverage

Unit Test Generators
- Randoop
- Evosuite

GUI Test Generators
- GUItar
- Exsyst

<table>
<thead>
<tr>
<th>Application</th>
<th>Randoop</th>
<th>Evosuite</th>
<th>GUItar</th>
<th>Exsyst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addressbook</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculator</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TerpPresent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TerpSpreadSheet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TerpWord</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
but can these ever be invoked from here?

testing may detect errors here...
test generation at the system level
anything that happens here is real

test generation at the system level
The Oracle
"Without specification, there are no bugs—only surprises"

Brian Kernighan
Specifications

```
_removeChild_

\( \Delta \text{XMLElement} \)
child? : XML\_ELEMENT

child? \( \in \) enumerateChildren
child? \( \neq \) null
enumerateChildren' = enumerateChildren \setminus \text{child}?
getChildrenCount' = getChildrenCount - 1
```

This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

\[ \text{XMLElement} \text{child} \]
\[ \text{XMLElement} \text{child} \]
\[ \text{XMLElement} \text{child} \]
\[ \text{XMLElement} \text{child} \]

Note how the specification captures two important preconditions not stated in the documentation—

1. \( \text{child} \) be a child of the target node
2. \( \text{child} \) be non-null

Both properties are essential for generating test cases, for instance. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

### 1d.3 State of the Art

#### 1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The `removeChild()` code indeed reveals some insights:

From this code, any static analysis can easily deduce:

- \( \text{child} \) be a child of the target node
- \( \text{child} \) be non-null

But how would one go about obtaining such specifications in practice?
Specifications

fully automated testing

diagram showing a specification for `removeChild` method:

```plaintext
 rencontChild
 ΔXMLElement
 child? : XML_ELEMENT

child? ∈ enumerateChildren
child? ≠ null
enumerateChildren' = enumerateChildren \ child?
getChildrenCount' = getChildrenCount − 1
```

fully automated debugging

widely automated verification

State of the Art

1d.3 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The `removeChild` code indeed reveals some insights: From this code, any static analysis can easily deduce precondition: $\text{child} \not\in \text{enumerateChildren}$.

But how would
(a) Executable Program
(b) Specification
(c) Test
fully automated debugging
fully automated testing
widely automated verification

Note how the specification captures two important preconditions not stated in the documentation— that $\text{child}$ be a child of the target node and that $\text{child}$ be non-null. Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.
Formal Methods for Dummies

Mark Zegarelli
Logic puzzle creator and expert

A Reference for the Rest of Us!

FREE eTips at dummies.com®
The Oracle

Microsoft Outlook

Unknown error

Was this information helpful?
Specifying Correctness

public class XMLElement implements IXMLElement, Serializable {

    // The name.
    private String name;

    // The child elements.
    private Vector children;

    // Returns an enumeration of all child elements.
    public Enumeration enumerateChildren() {
        // more methods and attributes...
    }

    // Returns the number of children.
    public int getChildrenCount() {
        // more methods and attributes...
    }

    // Removes a child element.
    public void removeChild(IXMLElement child) {
        // more methods and attributes...
    }
}

Figure 1: The XMLElement class from the NanoXML parser

This is precisely what our proposed approach produces: Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

```
removeChild(XMLElement child) =
    ^enumerateChildren = null
    ^enumerateChildren = enumerateChildren \ child
    ^getChildrenCount = getChildrenCount - 1
```

Figure 2: Mined specification for `removeChild` as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—`child` be a child of the target node and `child` be non-null. Both properties are essential for generating test cases. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art

1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The `removeChild()` code indeed reveals some insights: From this code, any static analysis can easily deduce precondition:

```
child = null
```

But how would (a) Executable Program (b) Specification (c) Test
Normality
Mining Normality

public class XMLElement implements IXMLElement, Serializable {
    private String name;
    private Vector children;

    public Enumeration enumerateChildren() {
        // Returns an enumeration of all child elements.
    }

    public int getChildrenCount() {
        // Returns the number of children.
    }

    public void removeChild(XMLElement child) {
        // Removes a child element.
    }
}

Figure 1: The XMLElement class from the NanoXML parser

This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild
XMLElement child:
XMLElement child ⇔ enumerateChildren(child) = null
enumerateChildren = enumerateChildren \ child
getChildrenCount = getChildrenCount − 1

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—child be a child of the target node and child be non-null. Both properties are essential for generating test cases, for instance. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties. The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:
child ⇔ = null

But how would
Facebook ermöglicht es dir, mit den Menschen in deinem Leben in Verbindung zu treten und Inhalte mit diesen zu teilen.
Normality: Platforms
Abnormal Behavior

Incorrect Dimensions: //div[@id='gus-wrapper']/div[1]/div

Reference Browser
Mozilla Firefox 17 / Windows XP

X-Browser
Internet Explorer 8 / Windows 7
Presence of *interactive* elements

Presence of *non-interactive* items

**Dimension** | **Position** | **Pixels**
webmate is a groundbreaking tool for the automatic testing of web applications, with a focus on cross-browser compatibility.
Normality: Time
webmate.io

webmate is a groundbreaking tool for the automatic testing of web applications, with a focus on cross-browser compatibility.

Public Beta November 2014
Normality: Apps
1. App collection

2. Topics

3. Clusters

4. Used APIs

5. Outliers
London Restaurants

looking for a restaurant, a bar, a pub or just to have fun in London? search no more! this application has all the information you need:

- you can search for every type of food you want: french, british, chinese, indian etc.
- you can use it if you are in a car, on a bicycle or walking
- you can view all objectives on the map
- you can search objectives
- you can view objectives near you
- you can view directions (visual route, distance and duration)
- you can use it with street view
- you can use it with navigation

keywords: london, restaurants, bars, pubs, food, breakfast, lunch, dinner, meal, eat, supper, street view, navigation
looking for a restaurant, a bar, a pub or just to have fun in London? search no more! This application has all the information you need:

- you can search for every type of food you want: French, British, Chinese, Indian etc.
- you can use it if you are in a car, on a bicycle or walking
- you can view all objectives on the map
- you can search objectives
- you can view objectives near you
- you can view directions (visual route, distance and duration)
- you can use it with street view
- you can use it with navigation

Keywords: London, restaurants, bars, pubs, food, breakfast, lunch, dinner, meal, eat, supper, street view, navigation.
Stemming

look london restaur search bar pub just applic fun
inform can search need every type food want french
british chines indian etc car bicycl walk
can us can view object map visual rout
can search object search can view distance
duracan view direct object near
can us street view can us navig
keyword london restaur bar pub food view
breakfast lunch dinner meal eat supper street navig
London Restaurant Topics

“navigation and travel” (59.8%)  
“food and recipes” (19.9%)  
“travel” (14.0%)  

look london restaur search bar pub just applic fun inform can search need everi type food want french british chines indian etc car bicycl walk  

keyword london restaur bar pub food view can search object search can view object map visual rout can us view direct object near durat can us view object map visual rout can us view direct object near  
breakfast lunch dinner meal eat supper street navig  

### Clusters

<table>
<thead>
<tr>
<th>Id</th>
<th>Assigned Name</th>
<th>Size</th>
<th>Most Important Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“sharing”</td>
<td>1,453</td>
<td>share (53%), settings and utils, navigation and travel</td>
</tr>
<tr>
<td>2</td>
<td>“puzzle and card games”</td>
<td>953</td>
<td>puzzle and card games (78%), share, game</td>
</tr>
<tr>
<td>3</td>
<td>“memory puzzles”</td>
<td>1,069</td>
<td>puzzle and card games (40%), game (12%), share</td>
</tr>
<tr>
<td>4</td>
<td>“music”</td>
<td>714</td>
<td>music (58%), share, settings and utils</td>
</tr>
<tr>
<td>5</td>
<td>“music videos”</td>
<td>773</td>
<td>popular media (44%), holidays and religion (20%), share</td>
</tr>
<tr>
<td>6</td>
<td>“religious wallpapers”</td>
<td>367</td>
<td>holidays and religion (56%), design and art, wallpapers</td>
</tr>
<tr>
<td>7</td>
<td>“language”</td>
<td>602</td>
<td>language (67%), share, settings and utils</td>
</tr>
<tr>
<td>8</td>
<td>“cheat sheets”</td>
<td>785</td>
<td>game and cheat sheets (76%), share, popular media</td>
</tr>
<tr>
<td>9</td>
<td>“utils”</td>
<td>1,300</td>
<td>settings and utils (62%), share, connection</td>
</tr>
<tr>
<td>10</td>
<td>“sports game”</td>
<td>1,306</td>
<td>game (63%), battle games, puzzle and card games</td>
</tr>
<tr>
<td>11</td>
<td>“battle games”</td>
<td>953</td>
<td>battle games (60%), game</td>
</tr>
<tr>
<td>Cluster</td>
<td>Topic</td>
<td>Size</td>
<td>Top 3 Topics</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>19</td>
<td>“sports”</td>
<td>580</td>
<td>sports (62%), share, popular media</td>
</tr>
<tr>
<td>20</td>
<td>“files and videos”</td>
<td>679</td>
<td>files and videos (63%), share, settings and utils</td>
</tr>
<tr>
<td>21</td>
<td>“search and browse”</td>
<td>363</td>
<td>search and browse (64%), game, puzzle and card games</td>
</tr>
<tr>
<td>22</td>
<td>“advertisements”</td>
<td>380</td>
<td>policies and ads (97%)</td>
</tr>
<tr>
<td>23</td>
<td>“design and art”</td>
<td>978</td>
<td>design and art (48%), share, game</td>
</tr>
<tr>
<td>24</td>
<td>“car games”</td>
<td>449</td>
<td>cars (51%), game, puzzle and card games</td>
</tr>
<tr>
<td>25</td>
<td>“tv live”</td>
<td>500</td>
<td>tv (57%), share, navigation and travel</td>
</tr>
<tr>
<td>26</td>
<td>“adult photo”</td>
<td>828</td>
<td>photo and social (59%), share, settings and utils</td>
</tr>
<tr>
<td>27</td>
<td>“adult wallpapers”</td>
<td>543</td>
<td>wallpapers (51%), share, kids and bodies</td>
</tr>
<tr>
<td>28</td>
<td>“ad wallpapers”</td>
<td>180</td>
<td>policies and ads (46%), wallpapers, settings and utils</td>
</tr>
<tr>
<td>29</td>
<td>“ringtones and sound”</td>
<td>662</td>
<td>ringtones and sound (68%), share, settings and utils</td>
</tr>
<tr>
<td>30</td>
<td>“theme wallpapers”</td>
<td>593</td>
<td>wallpapers (90%), holidays and religion, share</td>
</tr>
<tr>
<td>31</td>
<td>“personalize”</td>
<td>402</td>
<td>personalize (86%), share, settings and utils</td>
</tr>
<tr>
<td>32</td>
<td>“settings and wallpapers”</td>
<td>251</td>
<td>settings and utils (37%), wallpapers (37%), personalize</td>
</tr>
</tbody>
</table>
Outlier Analysis

- For each app, determine the APIs used
- In each cluster, identify outliers through one-class support vector machine (OC-SVM)
London Restaurants

→ An Outlier in the “Travel” Cluster

```java
android.net.ConnectivityManager.getActiveNetworkInfo()
android.webkit.WebView()
java.net.HttpURLConnection.connect()
android.app.NotificationManager.notify()
java.net.URL.openConnection()
android.telephony.TelephonyManager.getDeviceId()
org.apache.http.impl.client.DefaultHttpClient()
org.apache.http.impl.client.DefaultHttpClient.execute()
android.location.LocationManager.getBestProvider()
android.telephony.TelephonyManager.getLine1Number()
android.net.wifi.WifiManager.isWifiEnabled()
android.accounts.AccountManager.getAccountsByType()
android.net.wifi.WifiManager.getConnectionInfo()
android.location.LocationManager.getLastKnownLocation()
android.location.LocationManager.isProviderEnabled()
android.location.LocationManager.requestLocationUpdates()
android.net.NetworkInfo.isConnectedOrConnecting()
android.net.ConnectivityManager.getAllNetworkInfo()
```
CHABADA

1. App collection
2. Topics
   - "Weather"
   - "Map"
   - "Travel"
   - "Map"
   - "Theme"

3. Clusters
   - Weather + Travel
   - Themes

4. Used APIs
   - Internet
   - Access-Location

5. Outliers
   - Weather + Travel
   - Themes
   - Send-SMS

Malware recognition rate >80%
Information Flow

- Which sensitive APIs does the *device ID* flow to?
MUDFLOW

Malware recognition rate >86%

Training

Outlier Detection

Outlier Detector

\( d = 0.76 \)
Normality: Apps
Normality: Apps

connect to adserver.com

connect to exploit.com
Mining Normality

This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification. In the Z specification language, the mined specification for `removeChild()` is shown in Figure:

```
removeChild
XMLElement child
?:
XML_ELEMENT child
⇥ enumerateChildren
child
⇤ = null
enumerateChildren
0 = enumerateChildren \ child
getChildrenCount
0 = getChildrenCount
1
```

Figure :: Mined specification for `removeChild` as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—

- that `child` be a child of the target node
- that `child` be non-null

Both properties are essential for generating test cases for instance. The postconditions precisely describe the effect of `removeChild()` and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis
How does one obtain a specification like this?

Static analysis takes the program code and infers properties. The `removeChild` code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

```
child
⇥ = null
```

But how would
Úlfar Erlingsson
Mining Normality

public class XMLElement implements IXMLElement, Serializable {
    // The name.
    private String name;

    // The child elements.
    private Vector children;

    // Returns an enumeration of all child elements.
    public Enumeration enumerateChildren() {
        // more methods and attributes...
    }

    // Returns the number of children.
    public int getChildrenCount() {
        // more methods and attributes...
    }

    // Removes a child element.
    public void removeChild(XML_ELEMENT child) {
        // more methods and attributes...
    }

    Figure 1: The XMLElement class from the NanoXML parser

    This is precisely what our proposed approach produces:

    Given a program, we automatically produce a high-level specification.

    In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

    removeChild(XMLElement child) {
        enumerateChildren(child) = null
        enumerateChildren() = enumerateChildren() \ child
        getChildrenCount() = getChildrenCount() - 1
    }

    Figure 2: Mined specification for removeChild as set forth in this proposal

    Note how the specification captures two important preconditions not stated in the documentation—
    that child be a child of the target node and that child be non-null. Both properties are essential for
    generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties.

The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child ∈ enumerateChildren

But how would one automatically produce a high-level specification from this code? This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification.

In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild(XMLElement child) {
    enumerateChildren(child) = null
    enumerateChildren() = enumerateChildren() \ child
    getChildrenCount() = getChildrenCount() - 1
}

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—
that child be a child of the target node and that child be non-null. Both properties are essential for
generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties.

The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child ∈ enumerateChildren

But how would one automatically produce a high-level specification from this code? This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification.

In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild(XMLElement child) {
    enumerateChildren(child) = null
    enumerateChildren() = enumerateChildren() \ child
    getChildrenCount() = getChildrenCount() - 1
}

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—
that child be a child of the target node and that child be non-null. Both properties are essential for
generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties.

The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child ∈ enumerateChildren

But how would one automatically produce a high-level specification from this code? This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification.

In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild(XMLElement child) {
    enumerateChildren(child) = null
    enumerateChildren() = enumerateChildren() \ child
    getChildrenCount() = getChildrenCount() - 1
}

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—
that child be a child of the target node and that child be non-null. Both properties are essential for
generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties.

The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child ∈ enumerateChildren

But how would one automatically produce a high-level specification from this code? This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification.

In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild(XMLElement child) {
    enumerateChildren(child) = null
    enumerateChildren() = enumerateChildren() \ child
    getChildrenCount() = getChildrenCount() - 1
}

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—
that child be a child of the target node and that child be non-null. Both properties are essential for
generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties.

The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child ∈ enumerateChildren

But how would one automatically produce a high-level specification from this code? This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification.

In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild(XMLElement child) {
    enumerateChildren(child) = null
    enumerateChildren() = enumerateChildren() \ child
    getChildrenCount() = getChildrenCount() - 1
}

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—
that child be a child of the target node and that child be non-null. Both properties are essential for
generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.

1d.3 State of the Art
1d.3.1 Static Analysis

How does one obtain a specification like this? Static analysis takes the program code and infers properties.

The removeChild() code indeed reveals some insights:

From this code, any static analysis can easily deduce precondition:

child ∈ enumerateChildren

But how would one automatically produce a high-level specification from this code? This is precisely what our proposed approach produces:

Given a program, we automatically produce a high-level specification.

In the Z specification language, the mined specification for removeChild() is shown in Figure 2:

removeChild(XMLElement child) {
    enumerateChildren(child) = null
    enumerateChildren() = enumerateChildren() \ child
    getChildrenCount() = getChildrenCount() - 1
}

Figure 2: Mined specification for removeChild as set forth in this proposal

Note how the specification captures two important preconditions not stated in the documentation—
that child be a child of the target node and that child be non-null. Both properties are essential for
generating test cases. The postconditions precisely describe the effect of removeChild() and could be used as test oracles or as a base for program synthesis.