Pflichtenheft

Beste Gruppe

1. Dezember 2016

Inhaltsverzeichnis

1 Produktübersicht **6**
1.1 Die Syntax zur Angabe der formalen Eigenschaften 7

2 Zielbestimmung **9**
2.1 Musskriterien 9
2.2 Sollkriterien 10
2.3 Wunschkriterien 10
2.4 Abgrenzungskriterien 11

3 Produkteinsatz **12**
3.1 Anwendungsbereiche 12
3.2 Zielgruppen 12
3.3 Betriebsbedingungen 12

4 Produktumgebung **13**
4.1 Software 13
4.2 Hardware 13
4.3 Produkt-Schnittstellen 13

5 Funktionale Anforderungen **14**
5.1 Allgemein 14
5.1.1 Muss-Kriterien 14
5.1.2 Kann-Kriterien 14
5.2 C-Code Editor für Wahlverfahren 14
5.2.1 Muss-Kriterien 14
5.2.2 Soll-Kriterien 15
5.2.3 Kann-Kriterien 16
5.3 Editor für formale Eigenschaften 16
5.3.1 Muss-Kriterien 16
5.3.2 Soll-Kriterien 17
5.3.3 Kann-Kriterien 18
5.4 Eigenschaften-Liste 18
5.5 Editor für Eingabeparameter 18

6 Produktdaten **19**
6.1 Code-Editor Wahlverfahren 19
6.2 Editor von formalen Eigenschaften 19
6.3 Parameter 19

6.4 Projektdaten 19
6.5 Eigenschaften-Liste 19

7 Nichtfunktionale Anforderungen **20**

8 Globale Testfälle **21**
8.1 Testfälle für den C-Code Editor für Wahlverfahren 21
8.2 Testfälle für die Eigenschaften-Liste 21
8.3 Testfälle für den Editor für formale Eigenschaften 21
8.4 Testfälle für den Editor für Eingabeparameter 22

9 Systemmodelle **23**
9.1 Szenarien 23
9.2 Anwendungsfälle 24
9.3 High-Level-Beschreibung der Architektur 24

10 GUI **27**
10.1 C-Editor 27
10.2 Eigenschaften-Liste 32
10.3 Eigenschaften-Editor 34
10.4 Parameter Editor 35

11 Phasenverantwortliche **37**
11.1 Pflichtenheft 37
11.2 Entwurf 37
11.3 Implementierung 37
11.4 Qualitätssicherung 37
11.5 Abschlusspräsentation 37

Glossar **38**

Abbildungsverzeichnis

9.1 Skizzenhfte Beschreibung der Packete und ihrer Abhängigkeiten 25

10.1 Der C Editor ohne Code. Direkt unter dem Menü-Streifen befindet sich der Tool-Streifen 29
10.2 Der Dialog welcher dem User das Erstellen neuer Wahlverfahren ermöglicht 29
10.3 Der C Editor mit Code und Anzeige der Wahlart 30
10.4 Fehleranzeige bei syntaktischem Fehler ohne Maus-Hover (Kann-Kriterium) 30
10.5 Fehleranzeige bei syntaktischem Fehler mit Maus-Hover (Kann-Kriterium) 31
10.6 Fehleranzeige nach statischer Analyse 31
10.7 Eigenschaften-Liste vor einer Überprüfung 32
10.8 Eigenschaften-Liste während einer Überprüfung 32
10.9 Liste nach Überprüfung 33
10.10 Anzeige des Gegenbeispiels 33
10.11 Eigenschaften-Editor ohne Code mit symbolischen Variablen 34
10.12 Eigenschaften-Editor mit Beispielhafter Eigenschaft Anonymität und beispielhaft dargestelltem Syntax-Highlighting (Kann-Kriterium) 35
10.13 Der Parameter Editor. PERS steht für Professional Election Rigging System 36
10.14 Das Fenster, welches dem glsBenutzer erlaubt die an CBMC gereichten Argumente zu editieren 36

Abkürzungsverzeichnis

CBMC C Bounded Model Checker

BMC Bounded Model Checking

GUI Graphical User Interface

1 Produktübersicht

Wahlverfahren bilden den Grundstein unserer Demokratie. Dabei werden viele Anforderungen an sie gestellt, welche unsere intuitiven Ideen über Gerechtigkeit formalisieren: Proportionalität, Anonymität, etc. Moderne Wahlverfahren sind oft so komplex, dass sie viele überraschende und teils unerwünschte Eigenschaften haben. Nachweisen deren Abwesenheit ist absolut nicht trivial. So wurde beispielsweise 2008 das Bundestagswahlrecht vom BVerfG für verfassungswidrig erklärt, da es unter anderem die Gleichheit der Wirkung verschiedener Stimmen verletzte. Auf der anderen Seite ist es auch sehr schwer, Wahlverfahren auf die Präsenz erwünschter Eigenschaften zu untersuchen.

Bounded Model Checking (BMC) wird normalerweise dazu verwendet zu überprüfen, ob ein gegebenes Programm gegebene Eigenschaften erfüllt. Da dieses Problem im Allgemeinen unentscheidbar ist, werden nur endliche Codepfade überprüft. Dadurch wird der Zustandsraum endlich und das Problem entscheidbar. Um dies zu bewerkstelligen, werden potentiell unendliche Codepfade - also Schleifen - bis zu einer vom glsBenutzer bestimmten Grenze aufgerollt. Danach wird eine SAT-Formel erstellt, die erfüllbar ist, genau dann wenn das Programm einen Zustand einnehmen kann, welcher die gegebene Eigenschaft nicht erfüllt. Dies ist vollautomatisch und gibt bei Nichterfüllung das Gegenbeispiel zurück.

In unserem Fall kann BMC konkret dazu verwendet werden, ein C-Programm darauf zu untersuchen ob es im Falle gegebener Vorbedingungen gegebene Nachbedingungen erfüllt. Dies wird dazu verwendet, obige Problemstellung innerhalb einer bestimmten Genauigkeit zu lösen: so kann ein in C beschriebenes Wahlverfahren wie z.B. die einfache Mehrheitswahl darauf geprüft werden, ob es bestimmte Eigenschaften erfüllt. Allerdings ist es kompliziert, dies direkt zu tun.

Unser Programm ist im Wesentlichen eine sehr umfangreiche Schnittstelle um mit C Bounded Model Checker (CBMC) zu kommunizieren. Es bietet dem glsBenutzer über eine Graphical User Interface (GUI) die Möglichkeiten, formale Eigenschaften für Wahlverfahren sowie diese Wahlverfahren selbst anzugeben und zu editieren. Weiterhin liefert es Möglichkeiten, die Interaktion mit CBMC zu gestalten: Für wie viele Wähler, Plätze etc die Eigenschaft überprüft werden soll. Nach erfolgreicher Überprüfung durch CBMC bekommt der glsBenutzer schließlich eine Antwort des Programms, in der er bei Nichterfüllung der Eigenschaft ein Gegenbeispiel angezeigt bekommt. Wird kein Gegenbeispiel gefunden, so wird eine Erfolgsmeldung ausgegeben. All dies wird graphisch über die GUI aufbereitet.

Die GUI ist nach Funktionalität in vier Teilen angeordnet:

1. „C-Editor“: Code-Editor für Wahlverfahren in der Programmiersprache C
2. Ëigenschaften-Liste”: Listenansicht aller Eigenschaften, die für dieses Wahlverfah-

ren untersucht werden sollen

3. „Eigenschaften-Editor“: Editor für Spezifikation formaler Eigenschaften als boolsche Ausdrücke in eigens dafür vorgesehener Grammatik

4. „Params“: Eingabe von Parametern einer zu analysierenden Wahl

1.1 Die Syntax zur Angabe der formalen Eigenschaften

In diesem Abschnitt wird ein grober Überblick über die Sprache, welche der Eigenschaften-Editor verwendet, gegeben. Es handelt sich um ein Subset der C-Sprache mit einigen Ergänzungen. Diese werden im Folgenden erläutert.

Formale Eigenschaften werden in Vor- und Nachbedingungen unterteilt. Diese wiederum werden vom User als eine Liste boolscher Ausdrücke angegeben. Die Sprache erlaubt folgende Konstrukte zur Darstellung boolscher Ausdrücke:

- Folgende, aus C übernommene binäre Operatoren:

Operator	Symbol	Erwartet Argumente	Rückgabewert
Das logische Und	`&&`	zwei boolsche Ausdrücke	`true` oder `false`
Das logische Oder	`||`	zwei boolsche Ausdrücke	`true` oder `false`

- Folgende, zusätzlich hinzukommende binäre Operatoren:

Operator	Symbol	Erwartet Argumente	Rückgabewert
Die logische Implikation	`==>`	zwei boolsche Ausdrücke	`true` oder `false`
Die logische Äquivalenz	`<==>`	zwei boolsche Ausdrücke	`true` oder `false`

  Beispiel: `x > y <==> x + 1 > y + 1`
  Bedeutung: `x` ist größer als `y` genau dann, wenn auch `x + 1` größer als `y + 1` ist

- Folgende, aus C bekannte Operatoren. “Vergleichbarer Typen“ bedeutet, dass zwei Variablen dieser Typen bereits in C mit demselben Operator vergleichbar sind.

Operator	Symbol	Erwartet Argumente	Rückgabewert
Gleicheit	`==`	zwei Variablen vergleichbarer Typen	`true` oder `false`
Ungleicheit	`!=`	zwei Variablen vergleichbarer Typen	`true` oder `false`
kleiner als	`<`	zwei Variablen vergleichbarer Typen	`true` oder `false`
kleiner gleich	`<=`	zwei Variablen vergleichbarer Typen	`true` oder `false`
größer als	`>`	zwei Variablen vergleichbarer Typen	`true` oder `false`
größer gleich	`>=`	zwei Variablen vergleichbarer Typen	`true` oder `false`

- Symbolische Variablen vom Typ Wähler, Kandidat oder Sitz. Für deren Benennung gelten dieselben Regeln wie für die Benennung von Variablen in C.

Beispiel: `Wähler v, Kandidat c`

- Quantoren für Wähler, Kandidaten und Sitze in der Form von Makros. Ein bisher ungenutzter Variablenname wird als Argument erwartet. Dieser kann in dem darauf folgenden Ausdruck als symbolische Variable entsprechenden Typs verwendet werden.

  Beispiel: `FOR_ALL_VOTERS(v) : EXISTS_ONE_CANDIDATE(c) : v mag c`
  Bedeutung: Für jeden Wähler (`v`) gibt es zumindest einen Kandidaten (`c`) sodass das Tupel die Eigenschaft "der Wähler v mag den Kandidaten c" erfüllt (Dies ist nur ein Beispiel, der Editor wird die Eigenschaft 'mögen' nicht zu Verfügung stellen. An dessen Stelle könnte jedoch jeder andere binäre Operator stehen, welcher eine Variable vom Typ Wähler und eine Variable vom Typ Kandidat erwartet und `true` oder `false` zurückgibt).

- Ausgabe der Anzahl Stimmen für einen Kandidaten in der Form eines Makros.

- Viele Eigenschaften benötigen zu ihrer Überprüfung das Vergleichen mehrerer Wahldurchläufe. Dies wird ermöglicht durch Variablen VOTESx() und ELECTx. Dabei steht 'x' für die Nummer des Wahldurchangs. VOTESx() erwartet als Argument eine symblische Variable vom Typ Wähler. Zurück gibt es die Stimme, welche v im Wahldurchgang x abgegeben hat. ELECTx erwartet kein Argument und gibt das Wahlergebnis im x-ten Durchgang zurück. Der Rückgabetyp beider Makros hängt von der Kategorie des Wahlverfahrens ab. Gibt das Wahlverfahren zum Beispiel nur einen "Gewinner" aus, so ist ELECTx vom Typ Kandidat.

  Beispiel: `FOR_ALL_VOTERS(v) : VOTES1(v) == VOTES2(v)`
  Bedeutung: Alle Wähler wählen in beiden Wahlen (`VOTES1` und `VOTES2`) gleich.

  Beispiel: `ELECT1 == ELECT2` Bedeutung: Das Ergebnis des ersten Wahldurchlaufs stimmt mit dem des zweiten Wahldurchlaufs überein.

- Folgende Konstanten: Anzahl Wähler (`V`), Anzahl Kandidaten (`C`) und Anzahl Sitze. (`S`)

Beendet wird ein boolscher Ausdruck durch ein Semikolon.

2 Zielbestimmung

Ziel des Programmes ist es eine Lösung zur Untersuchung formaler Eigenschaften von Wahlverfahren zu präsentieren, welche auch von Nicht-Informatikern mit minimalem Aufwand erlernt und eingesetzt werden kann. Es soll Folgendes bereitstellen:

- Eine Möglichkeit zur Beschreibung von Wahlverfahren in C-Code
- Eine Möglichkeit zur Beschreibung der formalen Eigenschaften, welche das Wahlverfahren erfüllen soll, in der in 1.1 beschriebenen Syntax
- Eine Möglichkeit zum Angeben der Parameter (Anzahl Wähler, Anzahl Kandidaten, Anzahl Sitze)
- Eine Ausgabe des Ergebnisses der Überprüfung: eine Erfolgsmeldung bei Erfolg und Präsentation eines Gegenbeispiels bei Nichterfolg

Die Überprüfung der gegebenen Eigenschaften wird durch den CBMC geschehen. Aufgabe des Programmes wird es sein, die gegebenen Eingaben für den CBMC aufzubereiten, sowie dessen Ausgabe zu interpretieren und zu präsentieren.

All diese Aufgaben ließen sich theoretisch auch schon jetzt, ohne Verwendung unseres Programms erledigen. Allerdings wäre der damit verbundene Lern- und Einarbeitungsaufwand sehr hoch, vor allem bei der Angabe der formalen Eigenschaften. Weiterhin ist damit viel, sich jedes Mal wiederholender Aufwand, verbunden, welcher sich automatisieren lässt. Daher ist ein Schwerpunkt unseres Programmes einfache Benutzbarkeit, besonders für Nicht-Informatiker. Dies soll erreicht werden über eine GUI, welche oft benötigte Funktionalität bereitstellt. Einfache syntaktische Fehler im Code sollen während des Editierens erkannt werden. Dadurch soll das Untersuchen von Wahlverfahren leichter und schneller werden, was den Mehrwert unseres Programmes ausmacht.

2.1 Musskriterien

- Das Programm kann auf 32-Bit Versionen von Windows und Linux-Betriebssystemen betrieben werden
- Alle Abhängigkeiten werden mit dem Programm ausgeliefert
- Das Programm bietet einen Code-Editor für das zu prüfende Wahlverfahren
  - Der Code kann abgespeichert und geladen werden
  - Der Code-Editor zeigt Fehler im eingegebenen Code an

  - Aktionen können widerrufen und wiederhergestellt werden
- Es können formale Eigenschaften zur Prüfung des Wahlverfahrens eingegeben werden
  - Eine formale Eigenschaft kann abgespeichert und geladen werden
  - Fehler in der Eingabe werden angezeigt
- Die Parameter der Wahl (Anzahl von Wählern, Kandidaten und Sitzen) können festgelegt werden
- Das Ergebnis der Überprüfung wird vom Programm angezeigt. Im Fall der Verletzung einer formalen Eigenschaft wird ein Gegenbeispiel vom Programm angezeigt

2.2 Sollkriterien

- Die Parameter der Wahl können in Intervallen angegeben werden.
- Der Code-Editor bietet folgende Funktionalitäten:
  - Syntax-Highlighting
  - Automatisches Einrücken
  - Tastatur-Shortcuts
  - Codevorlagen
- Code completion bei der Eingabe der formalen Eigenschaften ist möglich
- Die Analyse des Wahlverfahrens kann abgebrochen werden

2.3 Wunschkriterien

- Das Programm kann auf einem Mac betrieben werden
- Der Code-Editor bietet folgende Funktionalitäten:
  - Code completion
  - Warnung vor nicht unterstützten Elementen der Programmiersprache wie z.B. Threads
- Es kann festgelegt werden, wie lange die Analyse des Wahlverfahrens maximal dauern soll
- Eine Wahl kann eingegeben werden. Das Ergebnis wird angezeigt

2.4 Abgrenzungskriterien

- Das Programm kann keine Angabe darüber machen, wie Lange die Überprüfung einer Eigenschaft dauern wird
- Es wird nicht bestätigt, ob ein gegebenes Wahlverfahren eine formale Eigenschaft erfüllt

3 Produkteinsatz

Das Programm überprüft Wahlverfahren auf ihre formalen Eigenschaften. Es richtet sich an Kunden, die ein Interesse an der Erforschung oder Entwicklung solcher Verfahren haben. Grundsätzlich sollte das Programm aber auch für Nicht-Informatikern verständlich sein. Für die Bedienung des Programms ist jedoch Kenntnis der Programmiersprache C und der Aussagen- und Prädikatenlogik nötig.

3.1 Anwendungsbereiche

- Universitärer Bereich
- Forschung

3.2 Zielgruppen

- Wahlforscher
- Softwareentwickler
- Hobbyisten

3.3 Betriebsbedingungen

Das Produkt kommt in einer Büroumgebung zum Einsatz. Es wird auf einem aktuellen Computer mit aktuellen Werten für Arbeitsspeicher und Rechnergeschwindigkeit betrieben.

4 Produktumgebung

4.1 Software

- Das Betriebssystem ist entweder Microsoft Windows (7 oder moderner) oder eine der Linux-Distributionen Arch oder Ubuntu lauffähig.

4.2 Hardware

- PC mit Tastatur und Maus

4.3 Produkt-Schnittstellen

Über das Produkt wird CBMC angesteuert.

5 Funktionale Anforderungen

5.1 Allgemein

5.1.1 Muss-Kriterien

/FM10/ Bereitstellen von Editoren zur Beschreibung des Wahlverfahrens sowie zur Beschreibung zu erfüllender formaler Eigenschaften
/FM20/ Kommunikation und Überprüfung dieser Eigenschaften via CBMC
/FM30/ Bereitstellen von Kommunikationsschnittstellen mit CBMC sowohl für Eingabe von Parametern als auch zur Ausgabe der Ergebnisse einer Überprüfung
/FM31/ Darstellung der Ergebnisse einer Überprüfung in für Nicht-Informatiker verständlicher Form:

- Zahlen in Dezimaldarstellung
- Ausgabe der Eingabeparameter des Wahldurchlaufs bzw. der Wahldurchläufe:
  - Ausgabe des Stimmen Arrays (Anzahl Stimmen und Stimmen)
  - Ausgabe der Kandidaten (Anzahl)
  - Ausgabe der Sitze (Anzahl)
- Ausgabe des Ergebnisses des Wahldurchlaufs bzw. der Wahldurchläufe

/FM40/ Möglichkeit des Speicherns von Code, formaler Anforderungen und Eingabeparametern als ein Projekt
/FM50/ Möglichkeit des Öffnens und Editierens der Projekte aus /F40/

5.1.2 Kann-Kriterien

/FK10/ Durch den Benutzer veränderbare Einstellungen wie verwendeter Font oder Font-Größe
/FK20/ Laden und Speichern der in /FK10/ genannten Einstellungen

5.2 C-Code Editor für Wahlverfahren

5.2.1 Muss-Kriterien

/FM10/ Darstellung aller für das Programmieren in C benötigten Zeichen
/FM20/ Veränderung des dargestellten Textes durch Eingabe über Tastatur und Maus

wie in Notepad
/FM30/ Speichern von erstelltem Code als Datei auf der Festplatte an vom User angegebenen Ort
/FM40/ Laden und Darstellen von Dateien korrekten Formats
/FM50/ Anzeigen syntaktischer Fehler in dem Fehler Fenster (siehe 10.6)

5.2.2 Soll-Kriterien

/FS10/ Überprüfen des Formats beim Ladevorgang. Falls falsches Format: Ausgabe einer Fehlermeldung
/FS20/ Syntax-Highlighting: Darstellung diverser Schlüsselwörter in anderen Farben als den Rest des Codes. Dies beinhaltet, ist jedoch nicht beschränkt auf:

- Typendeklaration (int, float, ...)
- Kontrollflow-Konstrukte (if, else, while...)
- Variablennamen
- Kommentare

/FS30/ Anzeigen der Zeilennummern am linken Zeilenrand
/FS40/ Anzeigen von Syntaktischen Fehlern im Code, welche durch einen Lexer oder Parser erkannt werden können:

- Verwendung von Schlüsselwörtern als Variablennamen
- Vergessene Semikolons am Ende von Anweisungen
- Nicht geschlossene Klammern und Anführungszeichen
- Andere Konstrukte, welche der Grammatik der C-Sprache widersprechen

/FS50/ Reaktion auf typische Tastenkürzel

Tabelle 5.1: Hotkeys und verbundene Operationen

Kürzel	Operation
Strg + c	Kopieren
Strg + x	Auschneiden
Strg + v	Einfügen
Strg + z	Zuletzt ausgeführte Aktion Rückgängig machen
Strg + r	Zuletzt rückgängig gemachte Aktion erneut ausführen
Strg + s	Speichern
Strg + o	Öffnen
Strg + Leer	Anzeigen der Code-Completion Vorschläge

/FS60/ Bereitstellen von Wahl-Templates

- Jeder Wähler wählt genau einen Kandidaten
- Jeder Wähler ordnet Kandidaten nach Präferenz in absteigender Reihenfolge
- Jeder Wähler ordnet Kandidaten eine Nummer zwischen MAX (maximale Zustimmung) und MIN (maximale Abneigung) zu. MAX und MIN sind dabei vom User konfigurierbar.

5.2.3 Kann-Kriterien

/FK10/ Automatisches Einrücken des Codes in Schleifen und if-Statements
/FK20/ Code-Completion

- Automatisches Schließen von Klammern und Anführungszeichen
- Primitiv: Vorschlagen bereits im Code vorgekommener Wörter
- Intelligent: Durch Analysieren eines ASTs nur Vorschlagen der Wörter welche im Kontext Sinn ergeben.

/FK30/ Durch den User konfigurierbares Verhalten:

- Festlegen der Farben, welche beim Syntax-Highlighting verwendet werden
- Festlegen des verwendeten Fonts
- An- und Ausschalten der angezeigten Zeilennummern
- Festlegen wie vielen Leerzeichen ein Tab entspricht

5.3 Editor für formale Eigenschaften

5.3.1 Muss-Kriterien

/FM10/ Darstellung aller für das Programmieren in C benötigten Zeichen
/FM20/ Veränderung des dargestellten Textes durch Eingabe über Tastatur und Maus wie in Notepad
/FM21/ Beschreibung von formalen Eigenschaften als Vor- und Nachbedingung /FM22/ Beschreibung der Vor- und Nachbedingungen als Auflistung boolscher Ausdrücke (siehe 1.1)
/FM30/ Bereitstellung von Makros zur Beschreibung der Eigenschaften (siehe 5.2)

Tabelle 5.2: Makros zur Beschreibung formaler Eigenschaften

Makro	Effekt
`FOR_ALL_VOTERS(i)`	In der darauf folgenden Eigenschaft kann i als symbolische Variable verwendet werden. Gesamtausdruck ist wahr falls sie für alle Wähler gilt
`FOR_ALL_CANDIDATES(i)`	In der darauf folgenden Eigenschaft kann i als symbolische Variable verwendet werden. Gesamtausdruck ist wahr falls sie für alle Kandidaten gilt
`FOR_ALL_SEATS(i)`	In der darauf folgenden Eigenschaft kann i als symbolische Variable verwendet werden. Gesamtausdruck ist wahr falls sie für alle Sitze gilt
`EXISTS_ONE_VOTER(i)`	In der darauf folgenden Eigenschaft kann i als symbolische Variable verwendet werden. Gesamtausdruck ist wahr falls sie für mindesten einen Wähler gilt
`EXISTS_ONE_CANDIDATE(i)`	In der darauf folgenden Eigenschaft kann i als symbolische Variable verwendet werden. Gesamtausdruck ist wahr falls sie für mindesten einen Kandidaten gilt
`EXISTS_ONE_SEAT(i)`	In der darauf folgenden Eigenschaft kann i als symbolische Variable verwendet werden.Gesamtausdruck ist wahr falls sie für mindesten einen Sitz gilt
`VOTE_SUM_FOR_CANDIDATE(c)`	Gibt die Anzahl Stimmen für Kandidaten c zurück

/FM40/ Bereitstellen symbolischer Variablen für Wähler, Kandidaten und Sitze
/FM50/ Bereitstellen von Operatoren für Implikation und Äquivalenz
/FM51/ Bereitstellen von Operatoren für logisches UND, ODER, GLEICH und NICHT GLEICH
/FM52/ Bereitstellen von Vergleichen: Kleiner, Kleiner gleich, größer, größer gleich, Gleicheit und Ungleicheit zwischen Typen die diese Vergleiche auch in C zulassen
/FM53/ Möglichkeit zur Abfrage der Stimme eines Wählers v in Wahldurchlauf Nummer x durch VOTESx(v) /FM54/ Möglichkeit zur Abfrage des Ergebnisses von Wahldurchlauf Nummer x durch ELECTx /FM60/ Beliebig tiefe, lediglich von Hardware begrenzte, Schachtelung dieser Konstrukte
/FM70/ Möglichkeit zum Speichern von Eigenschaften /FM70/ Möglichkeit zum Laden und Bearbeiten von Eigenschaften in korrektem Format

5.3.2 Soll-Kriterien

/FS10/ Syntax-Highlighting
/FS20/ Anzeigen von Syntaktischen Fehlern im Code

5.3.3 Kann-Kriterien

/FK10/ Code-Completion

- Auto-Vervollständigung der Makros
- Analyse des Codes und Anzeigen relevanter, bereits definierter Eigenschaften und symbolischer Variablen

5.4 Eigenschaften-Liste

/F10/ Darstellung der zu analysierenden Eigenschaften in Listenform /F20/ Möglichkeit zum Erstellen neuer Eigenschaften /F30/ Möglichkeit zum Hinzufügen bereits vorhandener Eigenschaften /F40/ Möglichkeit zum Entfernen von Eigenschaften /F50/ Möglichkeit das Überprüfen einzelner Eigenschaften an- und auszustellen /F60/ Möglichkeit Listen zu Speichern /F70/ Möglichkeit Listen in korrektem Format zu Laden

5.5 Editor für Eingabeparameter

/F10/ Möglichkeit zur Angabe der zu analysierenden Anzahl von Wählern, Kandidaten und Sitzen
/F21/ Möglichkeit die Parameter in /F10/ als Minimum und Maximum anzugeben, welche nacheinander abgearbeitet werden /F21/ Sanity-checks der eingegebenen Parameter: Alle größer 0, Minimum kleiner gleich Maximum /F20/ Möglichkeit zum Eingeben einer Zeitspanne nach welcher die Berechnung abgebrochen wird
/F30/ Möglichkeit direkter Eingabe von Argumenten, welche dem Aufruf von CBMC mitgegeben werden /F40/ Möglichkeiten das momentan geöffnete Wahlverfahren, die momentan eingestellten Parameter und die momentan geöffnete Eigenschaften-Liste als Projekt zu speichern /F50/ Möglichkeit die Projekte aus /F40/ zu öffnen und zu bearbeiten

6 Produktdaten

6.1 Code-Editor Wahlverfahren

/D10/ Das Wahlverfahren wird vom glsBenutzer als C-Code definiert und in einer Datei mit vom glsBenutzer gegebenen Namen und der Endung .c gespeichert. /D11/ Die verfügbaren Kategorien an Wahlverfahren werden in einer Datei namens electionTemplates gespeichert.

6.2 Editor von formalen Eigenschaften

/D20/ Eine vom glsBenutzer dafinierte formale Eigenschaft wird in einer Datei mit vom glsBenutzer gegebenen Namen und der Endung .eig gespeichert.

6.3 Parameter

/D30/ Angegebene Parameter für Wahlen werden in einer Textdatei gespeichert.

6.4 Projektdaten

/D40/ Ein Projekt wird als Liste von Dateien in einer Textdatei gespeichert.

6.5 Eigenschaften-Liste

/D50/ Die in der Eigenschaften-Liste erstellten Listen werden in Textdateien gespeichert

7 Nichtfunktionale Anforderungen

/F10/ Nicht mehr als 0,5 Sekunden Verzögerung bei Erfragen der Code-Completion

8 Globale Testfälle

8.1 Testfälle für den C-Code Editor für Wahlverfahren

/T110/ Erstellen eines neuen Dokuments
/T120/ Speichern des C-Codes
    /T121/ Speichern
    /T122/ Speichern unter
/T130/ Laden von C-Code aus einer Datei
/T140/ Auswahl des zu verwendenden Wahl-Templates
/T150/ Verwendung der Funktionen Ausschneiden, Einfügen, Kopieren, Rückgängig und Wiederholen
/T160/ Änderung der Eigenschaften des Editors
/T170/ Editieren des C-Codes
/T180/ Statische Analyse des Codes

8.2 Testfälle für die Eigenschaften-Liste

/T210/ Eine neue Liste erstellen
/T220/ Eine Liste speichern
    /T221/ Speichern
    /T222/ Speichern unter
/T230/ Eine Liste öffnen
/T240/ Ablesen des Ergebnisses der Überprüfung der formalen Eigenschaft.
/T250/ Einstellen ob eine formale Eigenschaft, die in der Liste geladen ist, in der nächsten Überprüfung verwendet werden soll.
/T260/ Die Funktionen Rückgängig und Wiederholen verwenden
/T270/ Eine neue formale Eigenschaft in die Liste aufnehmen

8.3 Testfälle für den Editor für formale Eigenschaften

/T310/ Eingabe und Editieren einer formalen Eigenschaft
    /T311/ Eingabe von Vor- und Nachbedingungen
    /T312/ Verwendung der Bereitgestellten Makros
    /T313/ Verwendung der symbolischer Variablen
/T320/ Eine Formale Eigenschaft speichern

/T321/ Speichern
/T322/ Speichern unter
/T330/ Laden einer formalen Eigenschaft aus einer Datei
/T340/ Verwendung der Funktionen Ausschneiden, Einfügen, Kopieren, Rückgängig und Wiederholen

8.4 Testfälle für den Editor für Eingabeparameter

/T410/ Eingabe der Anzahl von Wählern, Kandidaten und Sitzen
/T411/ Eventuell erfolgt die Eingabe in Intervallen
/T420/ Eingabe einer maximalen Zeitspanne
/T430/ Starten und stoppen der Überprüfung des Wahlverfahren auf die gewählte formale Eigenschaft(en)
/T440/ Eine komplettes Projekt speichern
/T441/ Speichern
/T442/ Speichern unter
/T450/ Eine komplettes Projekt laden
/T460/ Eine neues Projekt anlegen

9 Systemmodelle

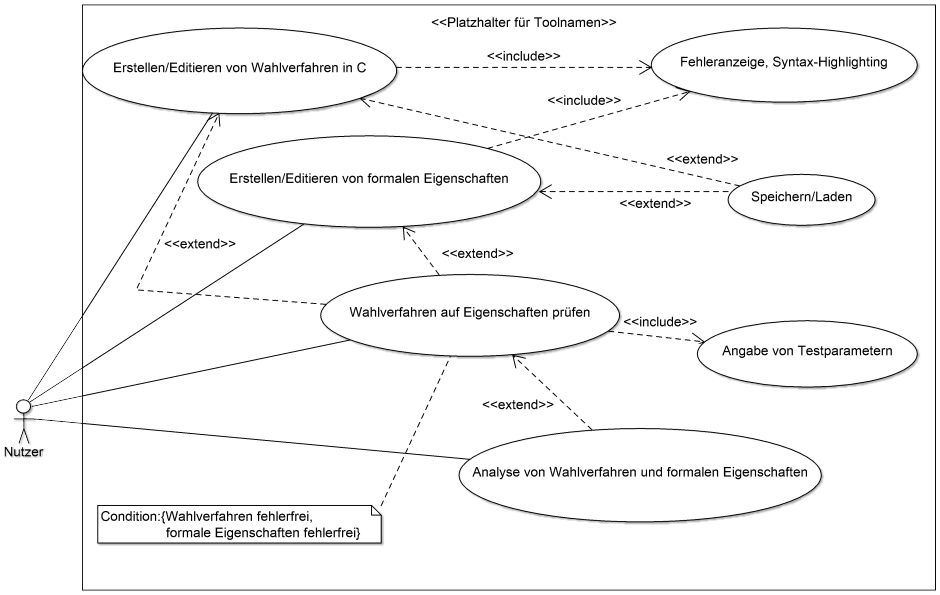
9.1 Szenarien

Szenario 1
Ein Wahlforscher wird von der Regierung eines Landes beauftragt ein neues Wahlverfahren zu entwerfen. Dieses soll bestimmte, vorgegebene formale Eigenschaften erfüllen. Er installiert das Tool und entwickelt mit diesem eine erste Version des Wahlverfahrens. Zusätzlich gibt er die vorgegebenen Eigenschaften als formale Eigenschaften in das Tool ein.
Nun gibt er Parameter zum Prüfen an (Wähler, Stimmen, Sitze) und lässt das Wahlverfahren auf die formalen Eigenschaften prüfen.
Das Tool gibt aus, dass nicht alle Eigenschaften vom Wahlverfahren erfüllt werden.
Mithilfe der Informationen aus der Ergebnisausgabe analysiert und modifiziert der Entwickler das Wahlverfahren so, dass es alle Eigenschaften erfüllt.
Er prüft es nochmals auf alle Eigenschaften und nun werden alle erfüllt.
Zuletzt speichert er Wahlverfahren und formale Eigenschaften ab und sendet ersteres an eine Prüfstelle für Wahlverfahren.

Szenario 2
Eine Prüfstelle bekommt ein Wahlverfahren übergeben welches sie auf bestimmte formale Eigenschaften testen soll.
Das Tool wird installiert und das Wahlverfahren im C-Editor geladen und es werden formale Eigenschaften erstellt.
Um das Wahlverfahren auf die Eigenschaften zu prüfen werden Parameter zu
Kandidaten-, Stimmen- und Wähleranzahl, sowie einem Timeout als obere Zeitgrenze für das Prüfen angegeben.
Wenn das Tool fertig mit dem Überprüfen ist, werden erfüllte und nichterfüllte Eigenschaften angezeigt. Diese Informationen benutzt die Prüfstelle dann in ihrem Bericht über das Wahlverfahren.

9.2 Anwendungsfälle

Der Nutzer (Entwickler, Wahlforscher, Prüfstelle..) kann Wahlverfahren und formale Eigenschaften in den jeweiligen Editoren des Tools entwickeln bzw. editieren. Beide Editoren verfügen über Optionen zum Laden und Speichern, sowie dem Anzeigen von Fehlern und Syntax-Highlighting.
Es kann geprüft werden ob das Wahlverfahren die formalen Eigenschaften erfüllt. Voraussetzung dafür sind ein fehlerfreies Wahlverfahren und fehlerfreie formale Eigenschaften. Vor dem Prüfen kann man Testparameter angeben (Anzahl von Kandidaten, Wählern, Sitzen und einem Timeout, dass dem Prüfen eine zeitliche Obergrenze gibt).
Nach dem Prüfen wird für jede Eigenschaft angezeigt ob sie vom Wahlverfahren erfüllt wird oder nicht. Werden formale Eigenschaften als nicht erfüllt erkannt, werden Gegenbeispiele angezeigt die die Nichterfüllbarkeit beweisen. Anhand dieser Informationen kann der Nutzer dann das Wahlverfahren und die Eigenschaften analysieren und falls nötig modifizieren.



9.3 High-Level-Beschreibung der Architektur

Es folgt eine skizzenhafte Beschreibung der High-Level-Architektur des Softwaresystems.

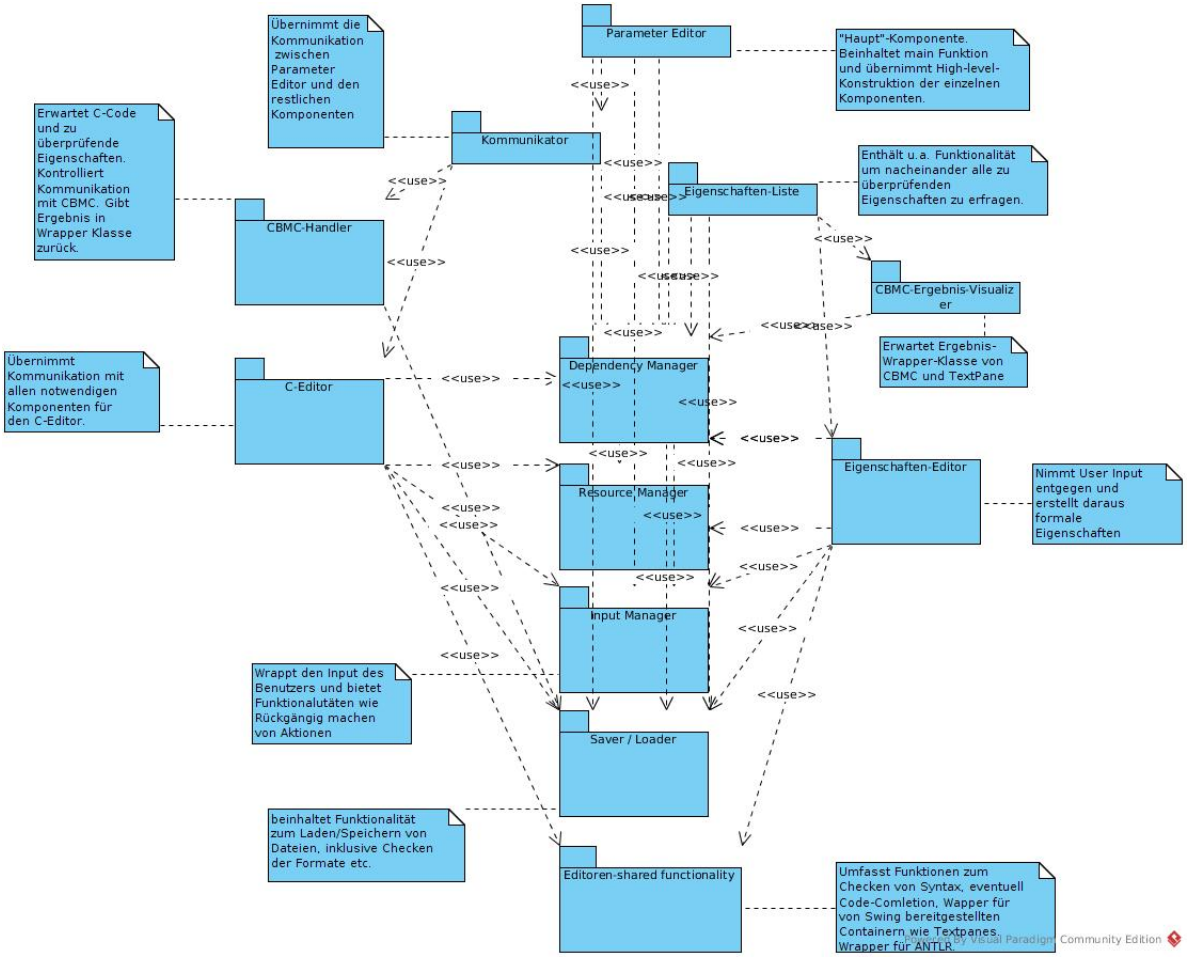


Abbildung 9.1: Skizzenhfte Beschreibung der Packete und ihrer Abhängigkeiten

9.1 zeigt einen ersten Entwurf der Packete und ihrer Abhängigkeiten untereinander. Diese einzelnen Packete sollen ähnliche Funktionalität bündeln. Zum momentanen Zeitpunkt liegen nur grobe Schätzungen vor wie aufwendig die implementierung der verschiedenen Funktionalitäten sein wird. Daher ist es möglich, dass manche Packete im Verlauf des Entwurfs als Klassen realisiert werden. Andernseits ist es auch möglich, dass weitere Packete hinzu kommen. Abhängigkeiten zu Java-Bibliotheken wie zum Beispiel Swing sind nicht dargestellt.

Ziel ist, möglichst viel gemeinsame Funktionalität zu kapseln und wiederzuverwerten. Beispielsweiße haben C- und Eigenschaften-Editor viele analoge Aufgaben. Beide müssen Text darstellen und es dem Benutzer erlauben, diesen zu bearbeiten. Beide müssen eingegebenen Code auf syntaktische Fehler untersuchen und die Ergebisse dieser Untersuchung dem Benutzer signalisieren. Unterschiede wird es geben in den Regeln der einzuhaltenden Syntax sowie der GUI. Daher wird diese Funktionalität gekapselt in ei-

nem eigenen Packet.

Diese Kapselung setzt voraus, dass die verschiedenen Komponenten mit den korrekten Abhängigkeiten konstruiert werden. Funktionalität hierzu befindet sich in dem Dependency-Manager. Dieser stellt eine Schnittstelle da über welche man Instanzen der benötigten Komponenten beziehen kann. In dem Diagram werden dennoch die konkreten "benutzen"Beziehungen visualisiert. In dem konkreten Entwurf werden diese jedoch über den Dependency Manager geregelt.

Das Kommunikator-Packet übernimmt die Kommunikation zwischen dem Parameter-Editor und den Komponenten. So wird es unter anderem der CBMC-Schnittstelle die korrekten Daten zukommen lassen und deren Ergebnisse an die Eigenschaften-Liste weiterleiten.

Alle Haupt-Komponenten müssen Dateien speichern und laden können. Ebenso müssen alle Haupt-Komponenten geladene Dateien auf korrektes Format hin überprüfen und das Ergebnis dieser Überprüfung mit dem Benutzer kommunizieren. Das mit Laden/Speichern beschriftete Packet wird Funktionalität zu diesem Zweck bereitstellen.

Die CBMC-Schnittstelle erwartet die Beschreibung des Wahlverfahrens sowie die zu überprüfenden Eigenschaften (jeweils in Wrapper-Klassen). Sie erzeugt den Input für CBMC, ruft CBMC mit diesem Input auf und gibt das Ergebnis (in einer Wrapper-Klasse) zurück.

Diese Wrapper-Klassen erleichtern den Umgang mit den enthaltenen Daten. Die Wrapper-Klasse für CBMC-Ergebnisse wird Funktionen zur Abfrage der Wahleingaben und Wahlergebnisse bereitstellen. Die Wrapper-Klassen für Eigenschaften und C-Code werden es erleichtern, den für CBMC notwendigen Code zu generieren. Zusätzlich erleichtern sie eine Trennung der internen (Model) und externen (View) Darstellung der Daten. So kann die Visualisierung der CBMC-Ergebnisse von einer weiteren Klasse übernommen werden. Diese erwartet lediglich die Ergebnis-Wrapper-Klasse sowie eine Swing-Komponente auf welcher das Ergebnis darzustellen ist. Allgemein sorgt diese zusätliche Abstraktion für eine flexible Architektur. Sollte das Program in Zukunft um weitere Programmiersprachen oder einen graphischen Editor zur Beschreibung von Wahlverfahren erweitert werden, so muss lediglich die entsprechende Wrapper-Klasse korrekt erzeugt werden.

Alle Editoren benötigen Funktionalität zum Wiederholen sowie Rückgängig machen von Aktionen. Dazu müssen Eingaben des Benutzers entsprechend gekapselt werden. Funktionaliät hierzu befindet sich im Input-Packet.

Sämtliche Resourcen wie Strings und durch den Benutzer konfigurierbare Eigenschaften wie der verwendete Font werden über den Resource Manager bereitgestellt. Dieser kann die vom Benutzer gegebenen Einstellungen auch speichern und laden.

10 GUI

Die hier vorgestellte GUI erfüllt alle Muss-, Soll- und Kann-Kriterien. Das endgültige Produkt kann daher davon abweichen. Im Folgenden wird jedes mal darauf hingewiesen, falls es sich bei einem Feature um ein Kann-Kriterium handelt. Die GUI besteht aus 4 verschiedenen Fenstern:

- Ein Editor für C-Code, in welchem die Wahlverfahren editiert werden können
- Eine Liste, in welcher alle für dieses Wahlverfahren zu überprüfenden Eigenschaften angezeigt werden
- Ein Editor in welchem Eigenschaften editiert werden können
- Das Hauptfenster, dessen Schließen ein Beenden des kompletten Tools nach sich zieht. Darin können Parameter für Überprüfungen eingestellt und Überprüfungen gestartet bzw. beendet werden

Jedes dieser Elemente verfügt auch über weitere Eigenschaften, die im Folgenden beschrieben werden.

10.1 C-Editor

Der C-Editor verfügt über dieselbe Funktionalität, welche andere Texteditoren wie zum Beispiel Notepad aufweisen. Ziel ist es, das Eingeben von Funktionen, welche ein Wahlverfahren implementiert, zu ermöglichen. Dazu bietet er die Möglichkeit, C-Code zu schreiben und zu bearbeiten. Ein angemessener Funktionskörper, welcher die auswählbare Art der Wahl, repräsentiert, wird dabei automatisch generiert (siehe 10.3). Es wird nicht möglich sein, diese Funktion zu editieren. Während des Eingebens des Codes wird dieser automatisch analysiert, um Schlüsselwörter sowie syntaktische Fehler zu markieren. Der C-Editor teilt sich in vier Untereinheiten auf: Der Menüstreifen, die Tool-Leiste, das Textfeld und das Fehlerfeld. Der Menü-Streifen ist unterteilt in Datei, Bearbeiten, Editor (Kann) und Code. Bilder aller geöffneten Untermenüs befinden sich im Anhang. Sie beinhalten folgende Funktionalität:

Menüpunkt	Bedeutung
Neu	öffnet ein neues Dokument, wobei die Art der Wahl vom User angegeben wird
Speichern	speichert das Dokument unter bereits gegebenem Namen
Speichern unter	Speichert das Dokument unter neuem Namen an neuem Ort, beide durch User angegeben
Öffnen	Öffnet neues Dokument des richtigen Formats

Tabelle 10.1: Unterpunkte des Datei-Menüs

Menüpunkt	Bedeutung
Rückgängig	Falls möglich: Macht die letzte ausgeführte Aktion Rückgängig
Wiederholen	Wiederholt die zuletzt Rückgängig gemachte Aktion
Kopieren	Fügt markierten Text in die Zwischenablage ein
Ausschneiden	Fügt markierten Text in die Zwischenablage ein und entfernt ihn aus dem Textfeld
Einfügen	Fügt Text aus der Zwischenablage an der Stelle des Cursors ein
Wahlart ändern	Ändert den Funktionskörper zu dem der vom User ausgewählten Art

Tabelle 10.2: Unterpunkte des Bearbeiten-Menüs

Menüpunkt	Bedeutung
Einstellungen	Öffnet den Einstellungen-Dialog. Dies ist Teil der Kann-Kriterien. Falls implementiert, wird es Möglichkeiten zur Einstellung des Fonts und Syntax-Highlighting geben.

Tabelle 10.3: Unterpunkte des Editor-Menüs

Menüpunkt	Bedeutung
Statische Analyse	Startet eine statische Analyse des Codes, welche ihn auf von Lexer oder Parser erkennbare Fehler untersucht. Gefundene Fehler werden in dem Fehlerfeld angezeigt. Zusätzliches Kann-Kriterium: Die Zeile, in welcher der Fehler ist, wird zusätzlich durch einen roten Punkt markiert (siehe 10.6

Tabelle 10.4: Unterpunkte des Code-Menüs

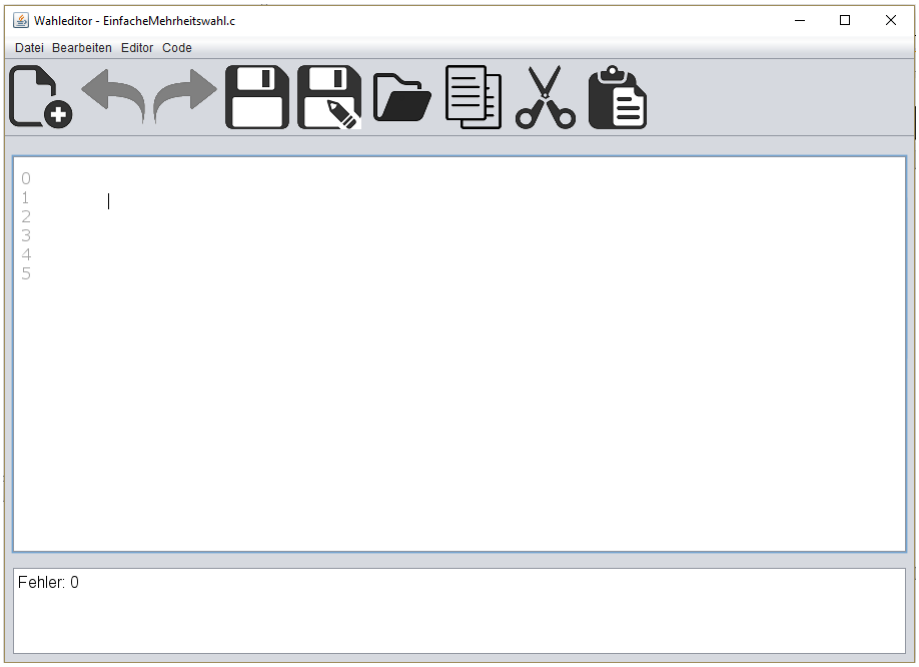


Abbildung 10.1: Der C Editor ohne Code. Direkt unter dem Menü-Streifen befindet sich der Tool-Streifen

Über den Tool-Streifen (siehe 10.2) lassen sich einige dieser Aktionen ohne Öffnen eines Menüs ausführen. Von Links nach Rechts: Neu, Rückgängig, Wiederholen, Speichern, Speichern unter, Öffnen, Kopieren, Ausschneiden, Einfügen. Aktivieren der Aktion "Neu" öffnet einen Dialog, welcher vim Nutzer die Eingabe folgender Eigenschaften erwartet: Speicherort, Name und Kategorie des Wahlverfahrens.

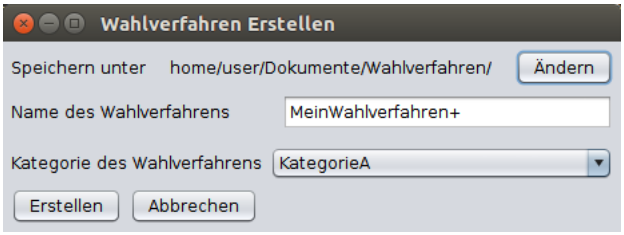


Abbildung 10.2: Der Dialog welcher dem User das Erstellen neuer Wahlverfahren ermöglicht

Bei erfolgreicher Eingabe wird eine neue Datei geöffnet, welche der Nutzer editieren kann. Die Kategorie des Wahlverfahrens wird durch die Funktionssignatur der Wahlfunktion (`voting`) dargestellt, welche nicht vom Nutzer verändert werden kann.

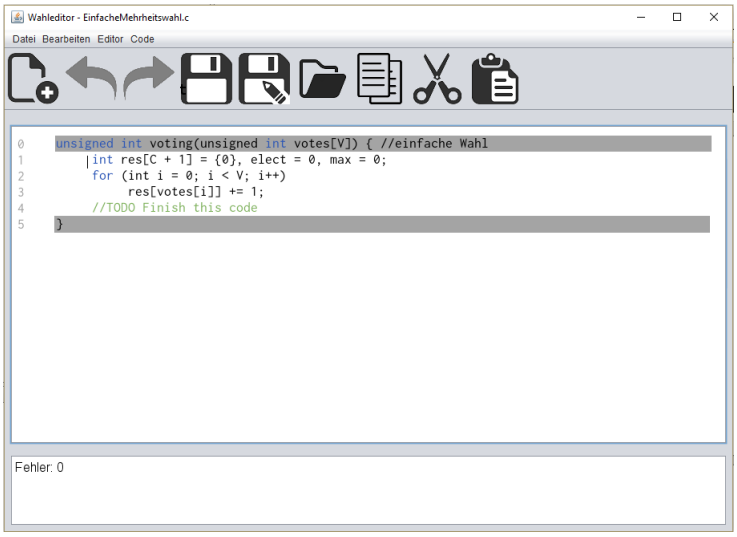


Abbildung 10.3: Der C Editor mit Code und Anzeige der Wahlart

In 10.3 sieht man den Editor nach Eingabe diverser Elemente der C-Sprache. Anzeige der Zeilennummern ist Soll-Kriterium, Möglichkeit diese Funktion zu deaktivieren Kann-Kriterium. Die grauen Balken zeigen an, dass man den vorgegebenen Funktionskörper nicht editieren kann. Der Kommentar rechts der Funktionsdefinition gibt die Kategorie des Wahlverfahrens an.

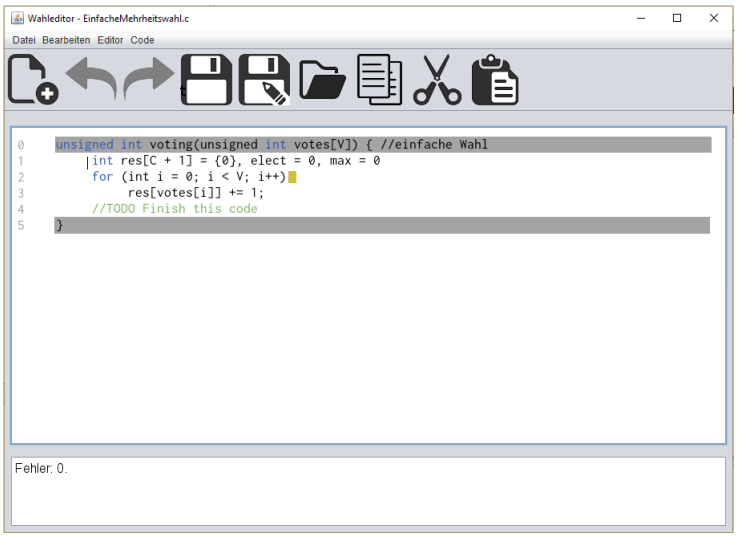


Abbildung 10.4: Fehleranzeige bei syntaktischem Fehler ohne Maus-Hover (Kann-Kriterium)

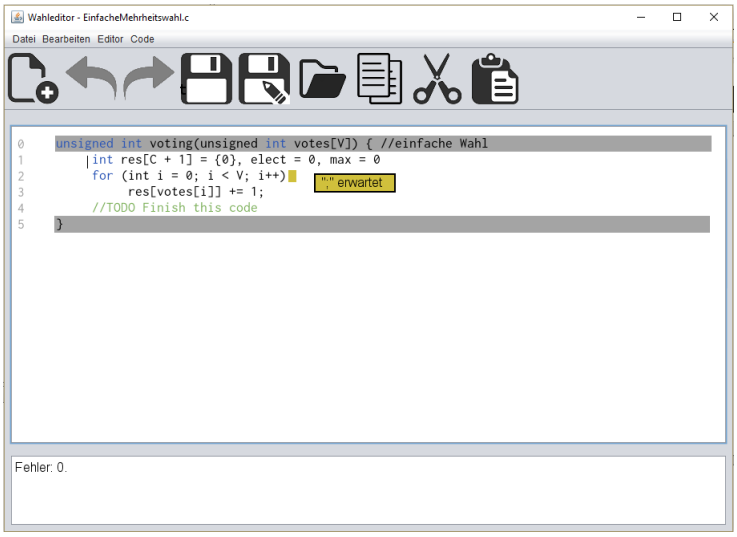


Abbildung 10.5: Fehleranzeige bei syntaktischem Fehler mit Maus-Hover (Kann-Kriterium)

In 10.4 sieht man, wie der gefundene syntaktische Fehler während des Editieren des Codes angezeigt wird. Dies ist ein Kann-Kriterium. Sobald man mit der Maus über die markierte Stelle geht, wird in einem neuen Fenster nahe der Maus eine Beschreibung des Fehlers angezeigt. Dies ist ebenfalls Kann-Kriterium (siehe 10.5).

Abbildung 10.6: Fehleranzeige nach statischer Analyse

10.6 Zeigt die Anzeige der Fehler nach ausführend einer statischen Code Analyse. Markierung der Zeile ist Kann-Kriterium.

10.2 Eigenschaften-Liste

Die GUI trennt das Editieren der zu überprüfenden Eigenschaften (in eigens zu diesem Zweck erstellter Syntax, siehe 1.1) und das Zuordnen dieser Eigenschaften zu Wahlverfahren. Dadurch können diese Eigenschaften einzeln abgespeichert und flexibel wiederverwertet und kombiniert werden. Das Zuordnen der Eigenschaften zu Wahlverfahren geschieht in der Eigenschaften-Liste. Darin werden die einzelnen Eigenschaften namentlich aufgelistet (siehe 10.7).

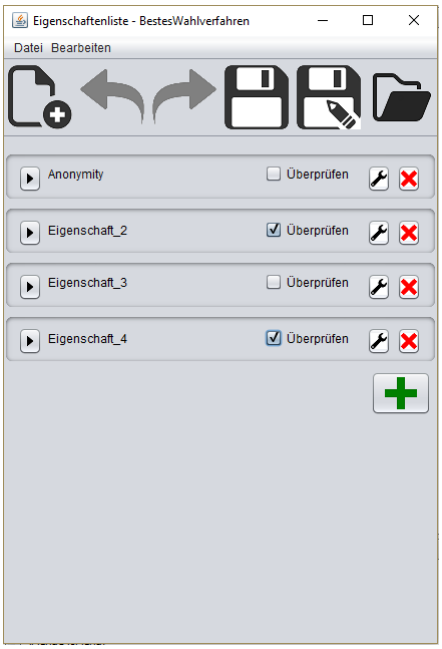


Abbildung 10.7: Eigenschaften-Liste vor einer Überprüfung

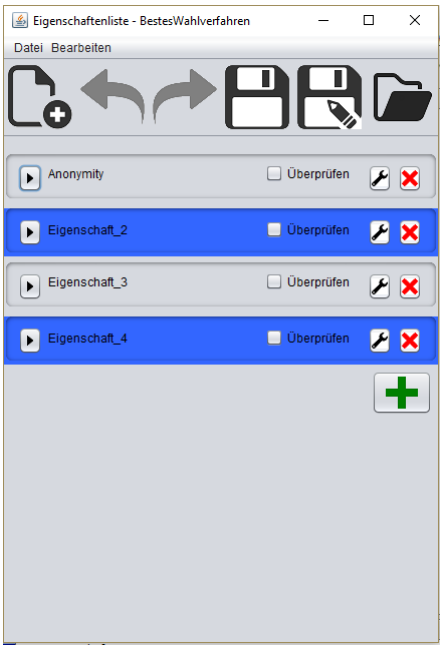


Abbildung 10.8: Eigenschaften-Liste während einer Überprüfung

Im Folgenden werden die einzelnen GUI-Bestandteile und der Funktionalität erläutert.

Menüpunkt	Bedeutung
Neu	Startet eine neue Liste
Speichern	Speichert die Liste
Speichern unter	Speichert die Liste unter neuem Namen an neuem Ort
Öffnen	öffnet Liste (Ort von User angegeben)

Tabelle 10.5: Unterpunkte des Datei-Menüs

Menüpunkt	Bedeutung
Rückgängig	Falls möglich: Macht die letzte ausgeführte Aktion Rückgängig
Wiederholen	Wiederholt die zuletzt Rückgängig gemachte Aktion

Tabelle 10.6: Unterpunkte des Bearbeiten-Menüs

Der Tool-Streifen hat exakt den selben Zweck wie der des Code-Editors, ohne die Funktionen Kopieren, Ausschneiden und Einfügen.

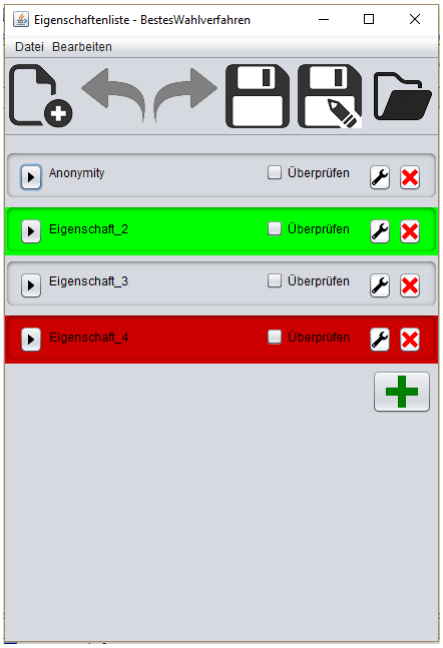


Abbildung 10.9: Liste nach Überprüfung

Abbildung 10.10: Anzeige des Gegenbeispiels

Es folgt eine Beschreibung der Icons, welche in den Listenelementen zu sehen sind.

Icon	Bedeutung
Pfeil nach rechts	Falls Gegenbeispiel gefunden: Durch Klicken öffnet sich unter dem Listenelement ein Textfeld in welchen das Gegenbeispiel dargestellt wird
Checkbox	Nur falls aktiviert, wird das Wahlverfahren auf die Eigenschaft getestet
Maulschlüssel	Öffnet den Eigenschaften-Editor für die Eigenschaft
Rotes Kreuz	Entfernt die Eigenschaft aus der Liste
Gründes Plus	Fügt neue, leere Eigenschaft oder bereits gespeicherte Eigenschaft der Liste hinzu

Tabelle 10.7: Icons der Eigenschaften-Liste

10.3 Eigenschaften-Editor

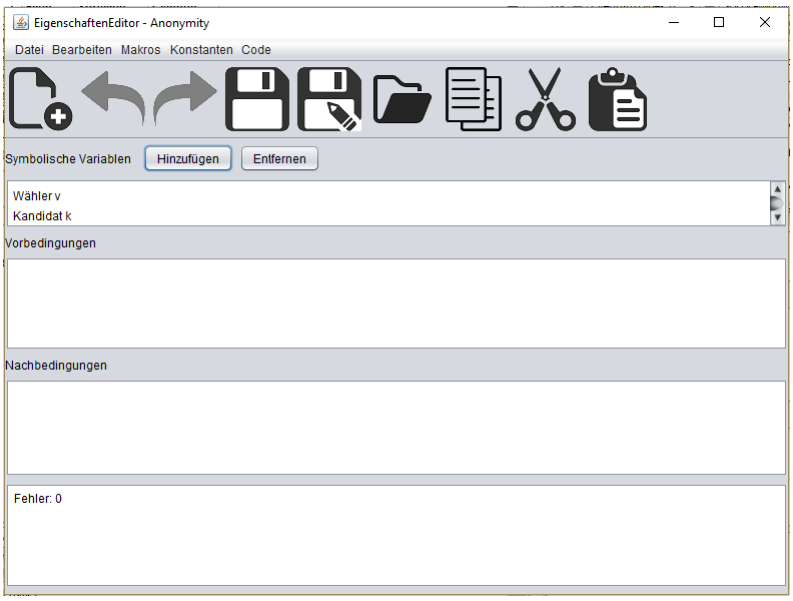


Abbildung 10.11: Eigenschaften-Editor ohne Code mit symbolischen Variablen

Der Eigenschaften-Editor hat den Zweck, das Bearbeiten der zu überprüfenden Eigenschaften zu ermöglichen. Eigenschaften werden aufgeteilt in Vor- und Nachbedingungen. Möglich ist die Verwendung sowohl von Quantoren in der Form von Makros als auch symbolischer Variablen.

Die Untermenüs Datei, Bearbeiten und Code sowie der Tool-Streifen sind analog zu denen des C-Editors. Hinzu kommen jedoch Konstanten und Makros. Bereitgestellte Konstanten sind:

- Die Anzahl der Wähler (V)
- Die Anzahl der Kandidaten (C)
- Die Anzahl vorhandener Sitze (S)

Bereitgestellte Makros sind:

- `FOR_ALL_VOTERS()`
- `FOR_ALL_CANDIDATES()`
- `FOR_ALL_SEATS()`
- `EXISTS_ONE_VOTER()`

- `EXISTS_ONE_CANDIDATES()`
- `EXISTS_ONE_SEAT()`
- `SUM_VOTES_FOR_CANDIDATE()`

Jedes dieser Makros bis auf das Letzte nimmt eine bisher ungenutzte Variable als Argument. Diese kann im darauf Folgenden boolschen Ausdruck verwendet werden (siehe 10.12). Das Letzte Makro nimmt eine bereits definierte symbolische Variable vom Typ Kandidaten und gibt berechnet die Anzahl Stimmen für diesen Kandidaten.

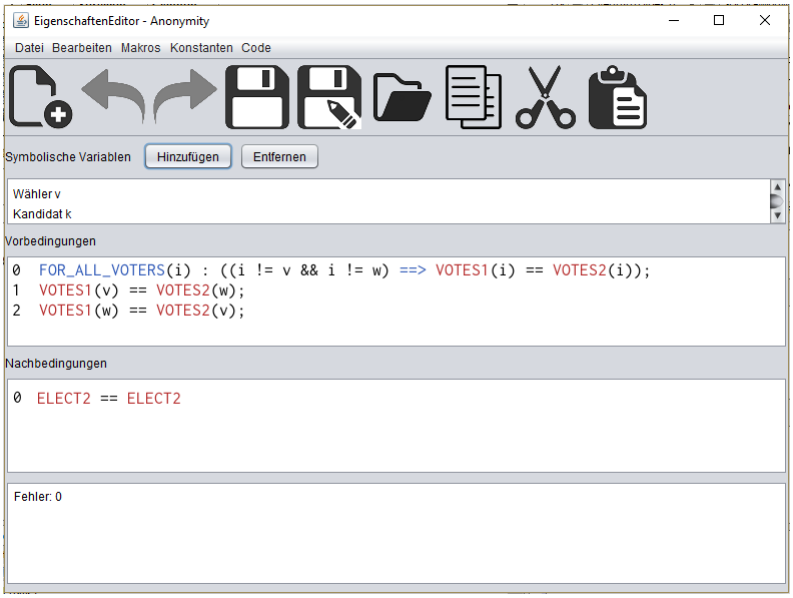


Abbildung 10.12: Eigenschaften-Editor mit Beispielhafter Eigenschaft Anonymität und beispielhaft dargestelltem Syntax-Highlighting (Kann-Kriterium)

Wie man zusätzlich sehen kann, sind auch Implikationen (`==>`) möglich. Auch das logische und (`&&`), oder (`||`) und die Äquivalenz (`<==>`) werden zur Verfügung stehen. Die Anzeige erkannter Fehler wird analog zum C-Editor geschehen.

10.4 Parameter Editor

Der Parameter Editor stellt das Hauptfenster der Anwendung dar. Es erlaubt das Einstellen und Starten der Tests. Hier lassen sich auch komplette Projekte Speichern - Code, Eigenschaften und Parameter gebündelt.

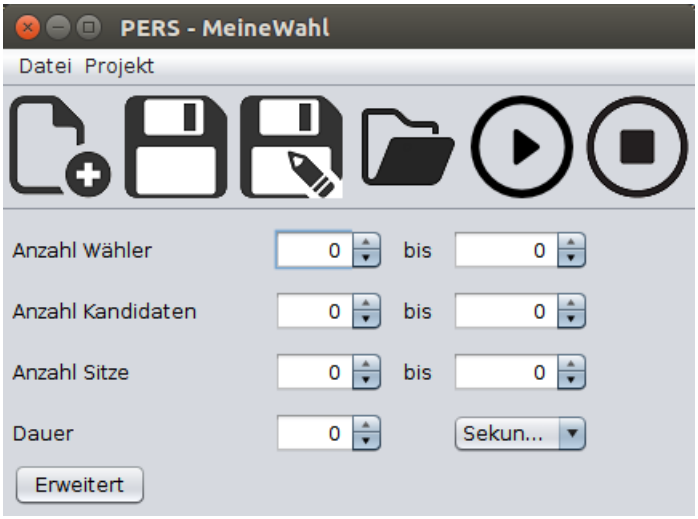


Abbildung 10.13: Der Parameter Editor. PERS steht für Professional Election Rigging System

Der Menüpunkt Datei ist analog zu dem des C Editors. Der Menüpunkt Projekt wird im Folgenden erläutert.

Menüpunkt	Bedeutung
Teste Eigenschaften	Startet Tests mit den angegebenen Parametern
Stop Test	Unterbricht momentan laufenden Test

Tabelle 10.8: Parameter Editor Projekt Menüpunkte

Der Pfeil und das Stopp-Zeichen des Tool-Streifens haben ebenfalls die Wirkungen Teste Eigenschaften und Stop Test. Als Mögliche Zeitangaben stehen Sekunden, Minuten, Stunden und Tage zur Verfügung.

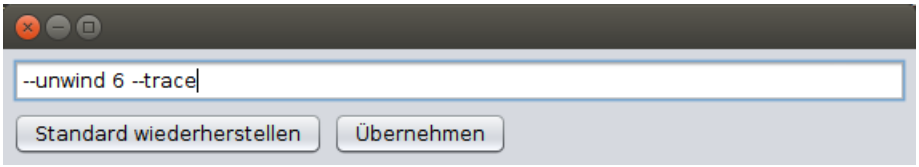


Abbildung 10.14: Das Fenster, welches dem glsBenutzer erlaubt die an CBMC gereichten Argumente zu editieren

Klickt der glsBenutzer auf Erweitert, so öffnet sich Dialog 10.14. Darin kann der glsBenutzer direkt die an CBMC gereichten Argumente editieren. Vom glsBenutzer gegebene Argumente unterliegen keiner Überprüfung durch das Program und führen daher zu undefiniertem Verhalten.

11 Phasenverantwortliche

11.1 Pflichtenheft

Justin Hecht

11.2 Entwurf

Holger Klein

11.3 Implementierung

Niels Hanselmann, Nikolai Schnell

11.4 Qualitätssicherung

Lukas Stapelbroek

11.5 Abschlusspräsentation

Jonas Wohnig

Glossar

Makro Ein Codeabschnitt, der durch den Präprozessor ersetzt wird.. 8