cp-logic

Umsetzung der Konzepte aus der Vorlesung *Diskrete Mathematik und Logik* in eine Toolbox fur Studierende (Bachelorarbeit)

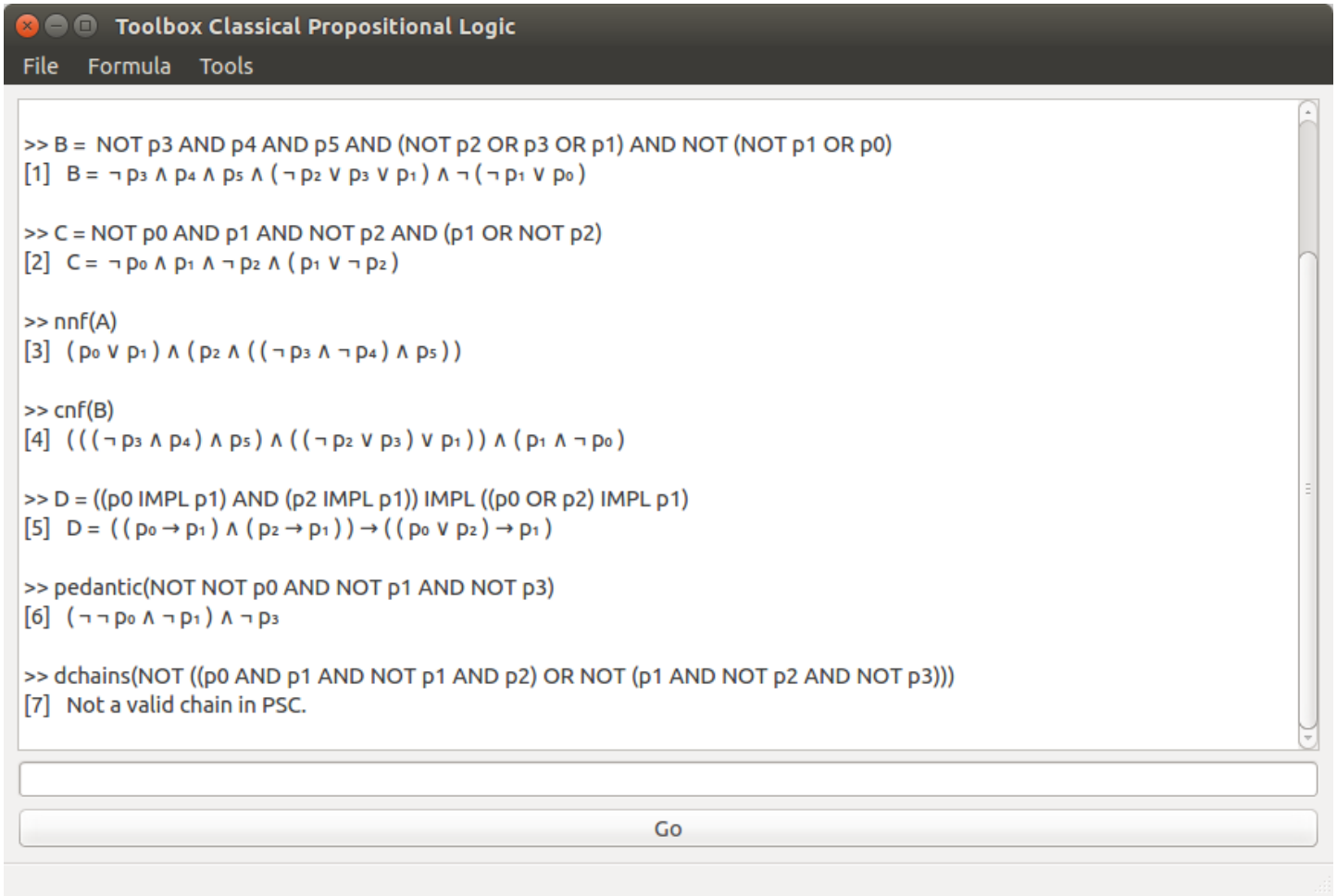
Adrianus Kleemans

22.05.2014

UNIVERSITÄT BERN

Inhaltsverzeichnis

GUI der Toolbox



Motivation

Ausgangspunkt: Skript **Diskrete Mathematik und Logik**
Im Skript werden viele Algorithmen beschrieben:

Definition 123 *The $\rightarrow$-reduction $\rho(A)$ of a formula A is inductively defined as follows:*

1. *If A is atomic, then $\rho(A) := A$.*
2. *If A is a formula $\neg B$, then $\rho(A) := \neg\rho(B)$.*
3. *If A is a formula $B \vee C$, then $\rho(A) := \rho(B) \vee \rho(C)$.*

Ideen:

- Ausprobieren durch Studierende
- Schnelles Feedback über Korrektheit

Anforderungen an das Endprodukt

- Nähe zum Skript
- Möglichst komplette Abbildung des Skripts
- Detaillierte Fehlerbehandlung
- Graphische Darstellung der Deduktionsketten
- Einfache Eingabemöglichkeit für Benutzer
- Laden von Files

Anforderungen an die Technologie

- Unicode-Unterstützung ($\vee$, $\wedge$, $\neg$, $\top$, ...)
- Geeignete Datenstrukturen (`[]`, `{}`)
- Unabhängigkeit vom Betriebssystem

Verwendete Technologien

Einige Verwendete Technologien:

- Programmiersprache: Python
- GUI: QT4, `pyside`, `pygame`
- Versionsverwaltung: git
- Unit tests: Modul `unittest`
- Unterstützende Technologien, z.B. `pyinstaller` zur Paketierung

Methoden

- TDD (Test Driven Development): Unit Tests
- Grundklasse: `Formula`, andere Klassen bauen darauf auf
-
- Interaktive Erarbeitung, Feedback Jannis

Test Driven Development

Zuerst die Spezifikation des Testfalls:

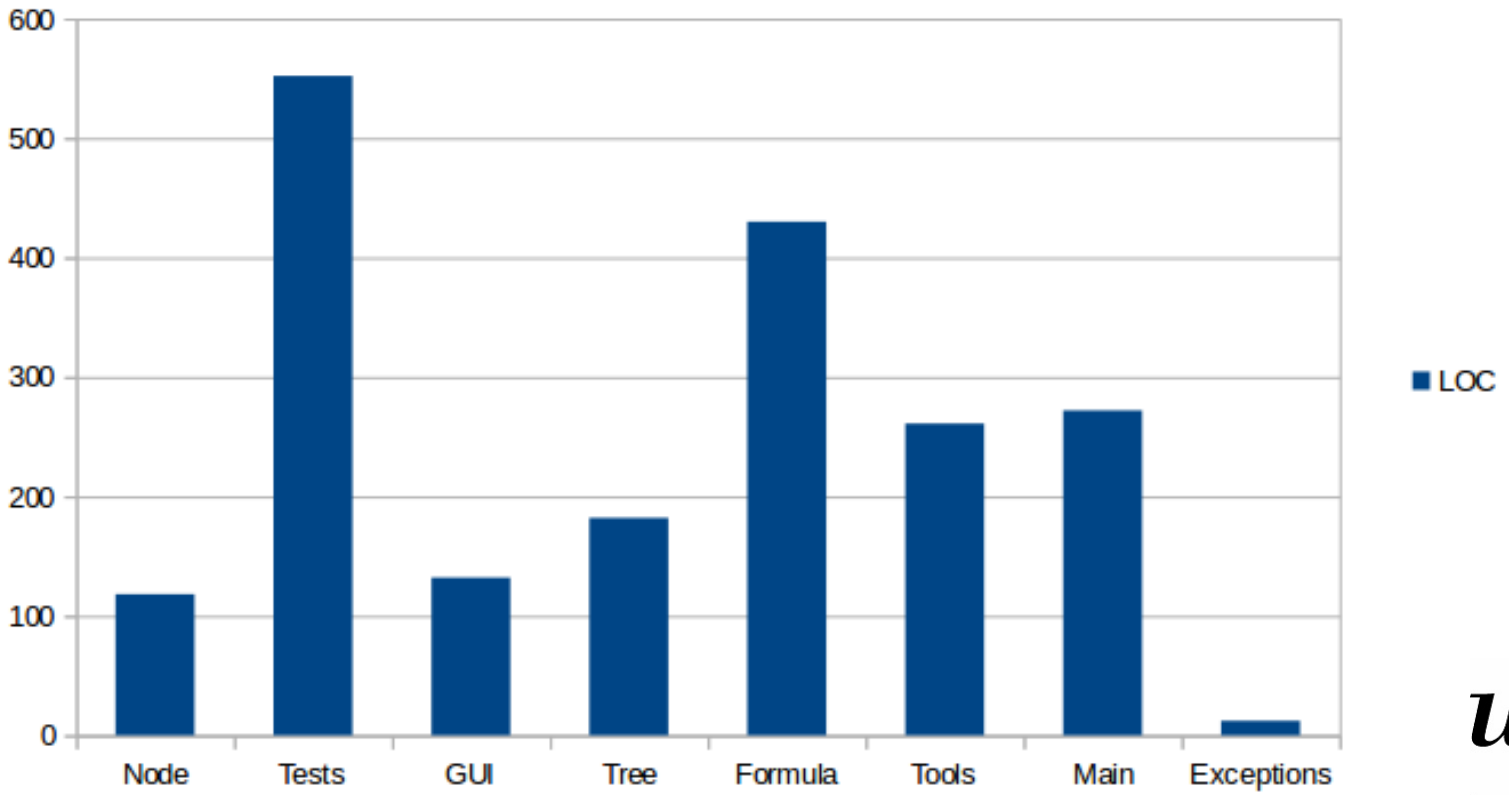
```
def test_invalidFormula_propositions(self):
    try: f = Formula('p0 OR OR NOT p1')
    except FormulaInvalidError: pass
    else: self.fail("Connectives without propositions
        are not allowed.")
```

Danach in der Formel-Klasse umsetzen, bis der Testfall erfolgreich durchläuft. $\Rightarrow$ Auch später noch Nutzen der Tests als Regressionstests!

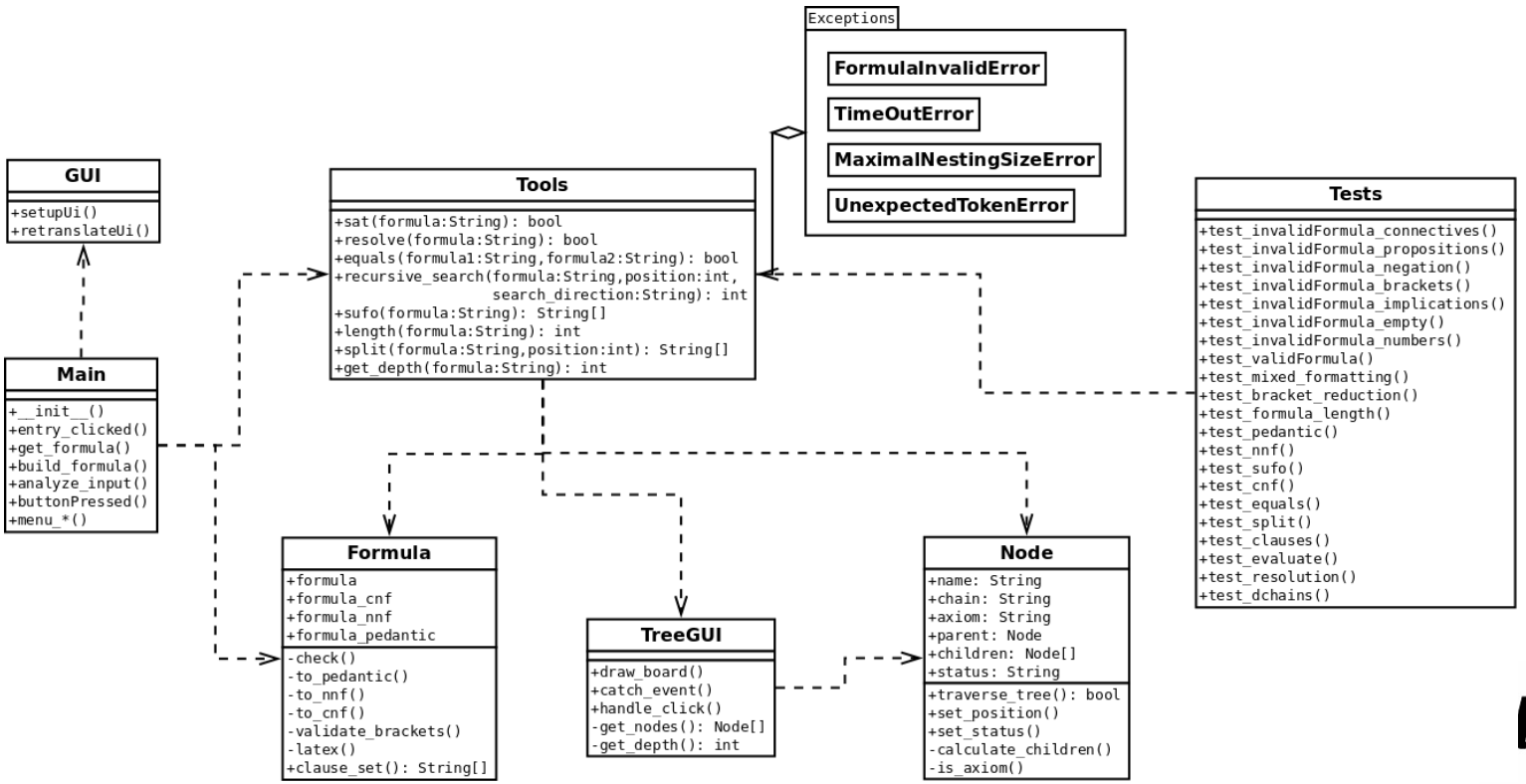
Ein paar Kennzahlen

- 8 Klassen mit insgesamt ~2000 Zeilen Code
- 35 Seiten Dokumentation für ~70 Funktionen
- insgesamt ~120 Tests, ein Grossteil für die Formel-Klasse
- Anforderungen umgesetzt

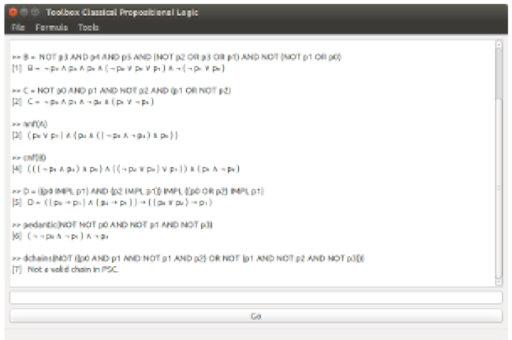
Klassen-Aufteilung

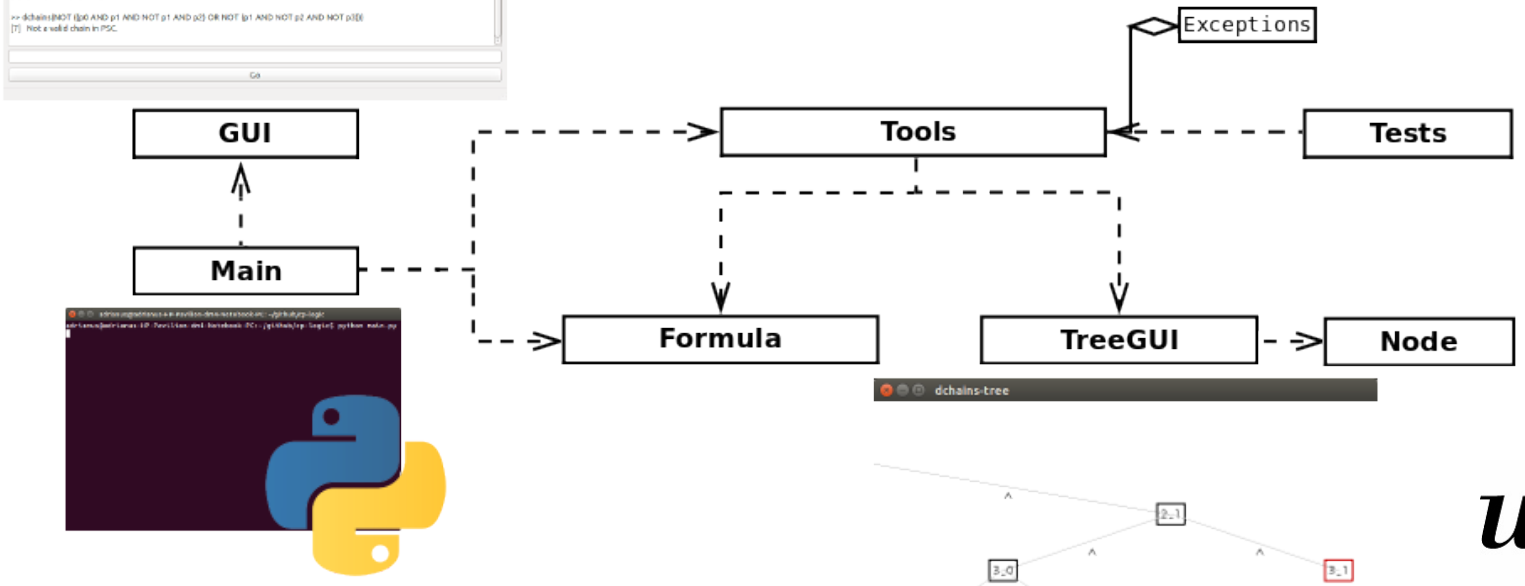


Aufbau


Exceptions
FormulaInvalidError
TimeOutError
MaximalNestingSizeError
UnexpectedTokenError
GUI
+setupUi()
+retranslateUi()
Tools
+sat(formula:String): bool
+resolve(formula:String): bool
+equals(formula1:String,formula2:String): bool
+recursive_search(formula:String,position:int,
search_direction:String): int
+sufo(formula:String): String[]
+length(formula:String): int
+split(formula:String,position:int): String[]
+get_depth(formula:String): int
Tests
+test_invalidFormula_connectives()
+test_invalidFormula_propositions()
+test_invalidFormula_negation()
+test_invalidFormula_brackets()
+test_invalidFormula_implications()
+test_invalidFormula_empty()
+test_invalidFormula_numbers()
+test_validFormula()
+test_mixed_formatting()
+test_bracket_reduction()
+test_formula_length()
+test_pedantic()
+test_nnf()
+test_sufo()
+test_cnf()
+test_equals()
+test_split()
+test_clauses()
+test_evaluate()
+test_resolution()
+test_dchains()
Main
+__init__()
+entry_clicked()
+get_formula()
+build_formula()
+analyze_input()
+buttonPressed()
+menu_*()
Formula
+formula
+formula_cnf
+formula_nnf
+formula_pedantic
-check()
-to_pedantic()
-to_nnf()
-to_cnf()
-validate_brackets()
-latex()
+clause_set(): String[]
TreeGUI
+draw_board()
+catch_event()
+handle_click()
-get_nodes(): Node[]
-get_depth(): int
Node
+name: String
+chain: String
+axiom: String
+parent: Node
+children: Node[]
+status: String
+traverse_tree(): bool
+set_position()
+set_status()
-calculate_children()
-is_axiom()

Aufbau


Toolbox Classical Propositional Logic
File Formula Tools
Go


Exceptions
GUI
Tools
Tests
Main
Formula
TreeGUI
Node

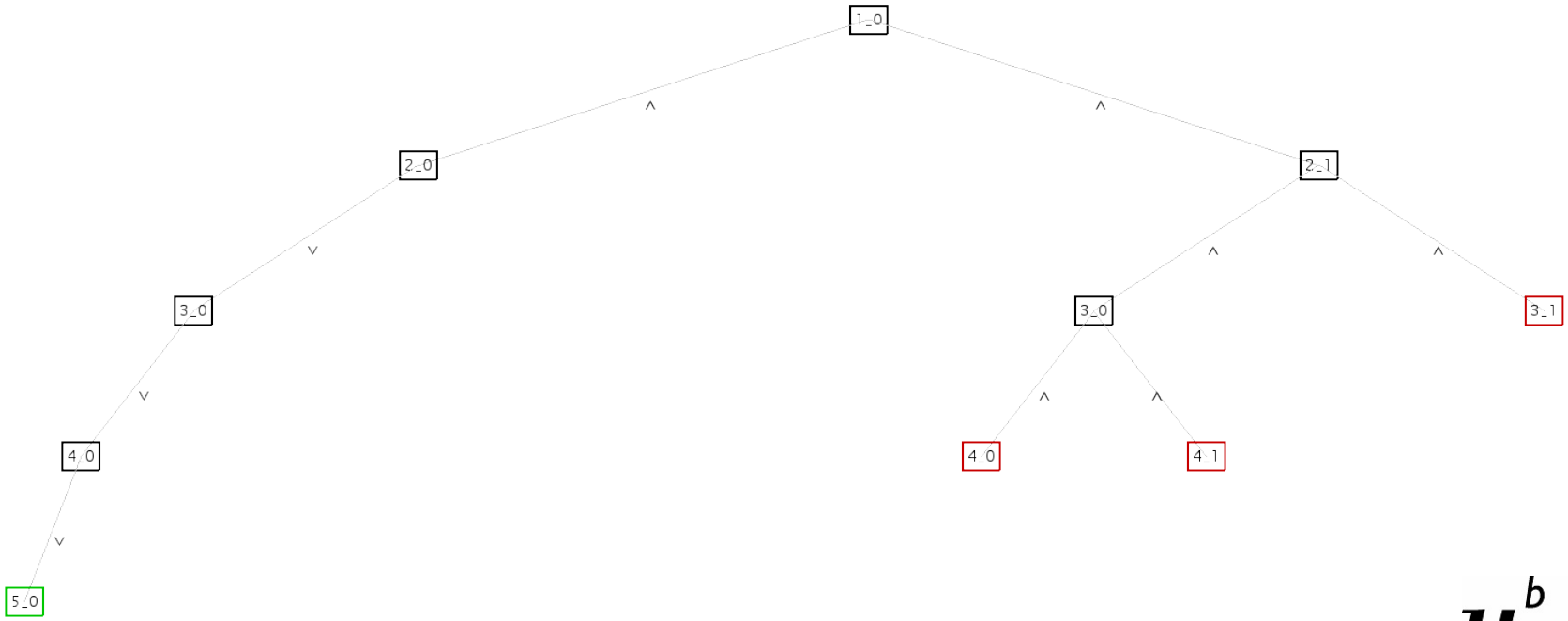
dchains-tree
2_1
3_0
3_1
4_0
4_1

Klassen

- **Main** - Einstiegspunkt
- **GUI** - Python/QT4-Benutzeroberfläche (Fenster mit Menü)
- **Formula** - Formel (mit Normalformen, Sufos, etc.)
- **Tools** - Weiterführende Funktionen (Erfüllbarkeit, Klauselmengen, lineare Suche, Tiefensuche etc.)
- **TreeGUI** - Deduktionsketten-Baum (Darstellung)
- **Node** - Deduktionsketten-Knoten (rekursiv berechnet)
- **Exceptions** - eigene Fehlerbehandlung
- **Tests** - Funktionale Tests, Regressionstests

Formula-Klasse

Deduktionsketten



Herausforderungen

- Unicode

Fazit

- Sehr viel gelernt (Logik, Zusammenspiel Technologie, GUI)
- Aufwand schwer abzuschätzen (Formel-Klasse vs. Klauselmengen)
- Umgehen mit Anforderungen, Änderungen, Prioritäten

Fragen?

Fragen?