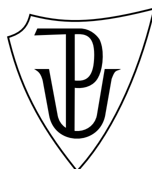
PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Grafické rozhraní pro SISC Scheme



2014 Filip Dušek

Anotace

Jazyk Scheme je jazyk, který si jistě zaslouží kvalitní a použitelné vývojové prostředí, bohužel najít prostředí, které by programátorovi poskytovalo dostatečnou míru pohodlí, není lehký úkol. Toto mě motivovalo k vytvoření grafického prostředí, nazvané Sisc Scheme, které má za cíl vyhnout se nedostatkům současně používaných nástrojů, co nejvíce programátorovi v jazyce Scheme usnadnit práci a posloužit jako alternativní nástroj pro výuku předmětů Paradigmata programování 1 a 2. Vytvořené prostředí využívá projekt SISC, který plně implementuje standart jazyka Scheme R5RS a zároveň přinaší i některá opravdu zajímavá rozšíření.

Děkuji Doc. RNDr. Vilému Vychodilovi, PhD. za jeho čas, rady a připomínky, které mě vždy nasměrovaly správným směrem.

Obsah

1. Úvod 7

1.1. Existující nástroje . . . 7

2. Projekt SISC 8

2.1. Základní rozšíření standardu R5RS projektem SISC . . . 8
2.1.1. Číselné konstanty . . . 8
2.1.2. Znakové konstanty . . . 8
2.1.3. Symboly . . . 8
2.1.4. Boxy . . . 8
2.1.5. Komentáře . . . 9
2.1.6. Podmíněné výrazy . . . 9
2.1.7. Vytváření vlastních specialních forem . . . 9
2.1.8. Prostředí . . . 9
2.1.9. Moduly . . . 10
2.2. Chyby a manipulace s nimi . . . 10
2.2.1. Manipulace s chybami během vyhodnocení . . . 10
2.2.2. Chybový záznam . . . 11
2.2.3. Vyvolání chyby . . . 11
2.3. Vyhodnocení výrazů jazyka Scheme v jazyce Java . . . 11
2.4. Používání jazyka Java z jazyka Scheme . . . 13
2.4.1. Převod elementů jazyka Scheme na objekty a zakladní typy jazyka Java . . . 13
2.4.2. Vytvoření nové instance . . . 13
2.4.3. Volání metody instance . . . 14
2.4.4. Přístup k proměnným instance . . . 14

3. Implementace aplikace 15

3.1. Balíček sisc.core.resources . . . 15
3.2. Balíček sisc.core . . . 15
3.3. Balíček sisc.engine . . . 15
3.4. Balíček sisc.inspector . . . 15
3.5. Balíček sisc.syntax . . . 16
3.6. Balíček gui.textpane . . . 17
3.7. Balíček gui.editor . . . 17
3.8. Balíček gui.interaktor . . . 17
3.9. Balíček gui.inspector . . . 17
3.10. Balíček gui.graph . . . 17
3.11. Balíček gui.debugger . . . 17
3.12. Balíček gui.settings . . . 17
3.13. Balíček gui.resources . . . 18
3.14. Balíček gui.application . . . 18

3.15. Známé problémy 18

Závěr **19**

Reference **20**

A. Příloha A: Uživatelská příručka **21**
A.1. Spuštění aplikace 21
A.2. Základní práce s editorem 21
A.3. Vyhodnocování výrazu uvnitř editoru 22
A.4. Otevření jednoho souboru ve více kartách 24
A.5. Práce s breakpointy 25
A.6. Práce s interaktorem 26
A.7. Inspektovaní struktur 27
A.8. Grafické zobrazení struktur 27
A.9. Nastavování klávesových zkratek 29

B. Obsah přiloženého CD **30**

Seznam obrázků

1. Editor. 21
2. Tlačítko pro vytvoření nové karty. 21
3. Tlačítko pro otevření souboru. 22
4. Dialog potvrzení otevření souboru. 22
5. Tlačítko pro uložení obsahu karty. 22
6. Tlačítko pro uložení souboru s novým jménem. 23
7. Tlačítko pro vrácení úprav. 23
8. Tlačítko pro znovuprovedení úprav. 23
9. Tlačítko pro vyhodnocení výrazu. 23
10. Tlačítko pro vyhodnocení posledního výrazu. 24
11. Tlačítko pro vyhodnocení celého obsahu karty. 24
12. Dialog změny v otevřeném souboru. 24
13. Tlačítko pro přidání breakpointu. 25
14. Seznam aktuálních breakpointů 25
15. Zobrazení aktuálního breakpointu. 26
16. Interaktor. 26
17. Inspektor. 27
18. Struktura zobrazená pomocí boxové notace s ukazateli. 28
19. Dialog pro nastavení klávesových zkratek. 29

1. Úvod

Programovací jazyk Scheme [8] je jeden z nejvýznamnějších dialektů jazyka Lisp. Scheme je minimalistický jazyk, který byl poprvé úplně popsán v roce 1978. Díky jednoduchosti své syntaxe je vhodný pro výuku programování, ale i pro vědecké výpočty.

1.1. Existující nástroje

Začínající programátor v jazyce Scheme má na výběr ze dvou zakladních typů rozhraní:

- grafické rozhraní např.: Racket [3] (dříve PLT Scheme, DrScheme)
- textové rozhraní např.: MIT/GNU Scheme [2], projekt SISC [5], projekt Schemik [4], projekt TinyScheme [7] atp.

Projektů zabývajícími se implementací jazyka Scheme s textovým rozhraním je na internetu k dispozici poměrně hodně. I když jsou některé z nich opravdu zajímavé, například implicitně paralení interpret z projektu Schemik [4], i přesto začínající programátor raději sáhne po grafické alternativě.

Bohužel právě v oblasti grafických prostředích není téměř žádná konkurence. Současný nejpoužívanější nástroj Racket [3] má spoustu funkcí, které se snaží zpříjemnit programátorovi práci, ale zároveň si s sebou nese i řadu nevýhod, například neumožňuje vyhodnocování výrazů v editoru po částech, ale vždy lze vyhodnotit jen celý obsah editoru jako celek. Toto sloužilo jako motivace pro vznik grafického rozhraní nazvaného Sisc Scheme nad projektem SISC [5], které se snaží poskytnout programátorům dostatečné pohodlí a vyhnout se hlavním nevýhodám nástroje Racket.

2. Projekt SISC

Projekt SISC (Second interpreter of Scheme code) [5] je mutliplatformní implementace jazyka Scheme v jazyce Java podle standardu R5RS [1] s rozšířeními užitečnými pro programování „skutečných“ aplikací [6]. Navíc projekt umožňuje kombinovat jazyky Scheme a Java pomocí modulu s2j (Scheme to Java). V této sekci rozeberu základní rozšíření projektu oproti standardu R5RS a na konci sekce ukážu jakym způsobem lze kombinovat jazyky Schema a Java.

2.1. Základní rozšíření standardu R5RS projektem SISC

2.1.1. Číselné konstanty

SISC přidává do jazyka číselné konstanty `#!+inf` (pozitivní nekonečno), `#!-inf` (záporné nekonečno), a `#!nan` (not a number). Tyto konstanty mohou vzniknou jako výsledek aritmetické operace například:

```
(/ 0 0.0) => #!nan
(/ 1 0.0) => #!+inf
(/ (- 1) 0.0) => #!-inf
```

2.1.2. Znakové konstanty

SISC přidává i nové znakové konstanty, znaky `#\space`(mezera) a `#\newline`(nový řádek).

2.1.3. Symboly

SISC nabízí vytváření symbolů, u kterých je rozpoznávána velikost písmen. Tyto symoboly musí začínat a končit znakem `|` (tzv. roura) viz následující ukázka:

```
(equal? 'foo '|foo|) => #t
(equal? 'Foo '|Foo|) => #f
```

2.1.4. Boxy

Boxy jsou speciální kontejnery pro hodnoty, se kterými lze manipulovat pomocí následujících procedur:

```
(box hodnota) => box
(unbox box) => value
(set-box! box hodnota) => nedefinovana hodnota
```

Procedura `box` vytvoří z předané hodnoty box obsahující hodnotu, procedura `unbox` naopak pro předaný box jako jediný argument vrátí hodnotu uvnitř boxu a procedura `set-box!` nastaví hodnotu uvnitř boxu na svůj druhý argument.

SISC samozřejmě poskytuje i predikát `box?` pro test, zda je předaný argument box.

2.1.5. Komentáře

SISC rozšiřuje i systém komentářů jazyka Scheme. K jednořádkovému komentáři začínajícího znakem `;` přidává komentář začínající znaky `#;` a více řádkový komentář, podobný jazyku Common Lisp, začínající znaky `#|` a ukončeným znaky `|#`.

2.1.6. Podmíněné výrazy

Podobně jako v jazyce Common Lisp, tak i v projektu SISC jsou k dispozici speciální formy `when` a `unless` s následující syntaxí:

```
(when podmínka výraz1 [výraz2] ... [výrazN])
(unless podmínka výraz1 [výraz2] ... [výrazN])
```

2.1.7. Vytváření vlastních specialních forem

Kromě speciální formy pro vytváření vlastních speciálních forem `define-syntax` ze standardu R5RS nabízí SISC nové speciální formy `define-macro` a `defmacro` s následující syntaxí:

```
(define-macro (jméno-formy . [argument1] ... [argumentN])
    tělo1 [tělo2] ... [těloN])
(define-macro jméno-formy expanzní-funkce)
(defmacro jméno-formy seznam-argumentů tělo1 [tělo2] ... [těloN])
```

2.1.8. Prostředí

V projektu SISC jsou prostředí elementy prvního řádu, lze je použít i jako argument procedury eval.

Pro vytvoření nového prostředí lze použít proceduru, která jako svůj jediný argument vyžaduje prostředí, které se použije jako rodič nově vzniklého prostředí. Napríklad:

```
(define env (make-child-environment (scheme-report-environment 5)))
env => #<environment env>
```

Z každého prostředí lze získat jeho předka pomocí procedury `parent-environment`, která bere jako svůj jediný argument prostředí.

Například.:

```
(parent-environment env) => #<environment>
```

Pro ukládání a získávání hodnot do nebo z prostředí nabízí SISC procedury `putprop` a `getprop` se syntaxí:

```
(putprop symbol prostředí hodnota)
(getprop symbol prostředí [výchozí-hodnota])
```

Použít výchozí hodnotu v proceduře `getprop` je vhodné, pokud chceme například otestovat jestli prostředí obsahuje hodnotu `#f`, protože procedura v případě že hodnotu v prostředí nenalezne vrací `#f`.

2.1.9. Moduly

Moduly umožnují programátorovi definici procedur s rozsahem pouze v oblasti definované modulem, podobné jmeným prostorům (namespace) nebo slotům a metodám objektů z jinych jazyků.

Modul vznikne použitím speciální formy module se syntaxí:

```
(module jméno-modulu
        ([exportovaný-symbol1] ... [exportovaný-symbolN])
        definice-symbolu1
        ⋮
        definice-symboluN)
```

Při použití modulu jsou viditelné pouze symboly uvedené v seznamu symbolů pro export (druhý argument specialní formy module). Pro použítí modulu je nutné nejdříve vyhodnotit speciální formu module a poté nově vzniklý modul importovat pomocí procedury `import`, to lze provést takto:

```
(import jméno-modulu) => #!void
```

2.2. Chyby a manipulace s nimi

2.2.1. Manipulace s chybami během vyhodnocení

Pro manipulaci s chybami během vyhodnocení výrazu slouží speciální forma with-failure-continuation s následující syntaxí:

```
(with-failure-continuation handler procedura-k-vyhodnocení)
```

Pro proceduru je k dispozici i její syntaktický cukr: `with/fc`. Procedura `with-failure-continuation` přijímá dva argumenty – dvě procedury.

Druhá procedura nesmí vyžadovat žádný argument a v případě, že její vyhodnocení proběhne bez chyby, tak je výsledek její aplikace vrácen jako výsledek aplikace procedury `with-failure-continuation`.

První procedura vyžaduje dva argumenty, které v případě, že aplikace druhé proběhne s chybou, obsahují popis chyby a kontext.

2.2.2. Chybový záznam

Chybový záznam (error record) slouží k uložení informací o chybě (chybová zpráva, místo vzniku, atp.) Pro vytvoření vlastního záznamu o chybě slouží procedura make-error se syntaxí:

```
(make-error [místo-vzniku] [zpráva] [argument1] ... [argumentN]) =>
  error-record
(make-error [místo-vzniku] hodnota) => error-record
```

Proceduru lze použít například takto:

```
(make-error 'misto "chybova zprava") =>
  ((message . "chybova zprava") (location . misto))
(make-error 'fib "Chyba behem ~a s argumentem ~a" 'fib 11) =>
  ((message . "Chyba behem fib s argumentem 11")
   (location . fib))
```

Pro přístup k jednotlivým položkám chybového záznamu slouží procedury `error-location` a `error-message`, obě berou jako svůj jediný argument chybový záznam. Procedura `error-location` vrací místo vzniku chyby a procedura `error-message` vrací chybovou zprávu.

2.2.3. Vyvolání chyby

K vyvolání chyby slouží procedura `throw` se syntaxí:

```
(throw chybový-záznam [chybové-pokračování]) => nic
```

Procedura při své aplikaci zobrazí chybu na standartní výstup, ale jako svůj výsledek nevrací žádnou hodnotu, viz příklad:

```
(throw (make-error 'fib "Chyba behem ~a s argumentem ~a" 'fib 11.1))
-> Error in fib: "Chyba behem fib s argumentem 11.1"
```

2.3. Vyhodnocení výrazů jazyka Scheme v jazyce Java

Pro vyhodnocení výrazu jazyka Scheme v jazyce Java potřebujeme samozřejmě soubor sisc.jar, který máme připojen jako knihovnu do vlastního projektu a soubor s haldou (`sisc.shp`). Nejdřív musíme vytvoři kontext aplikace a nahrát do něj soubor s haldou. Halda obsahuje kompilovaný kód nutný pro samotný běh interpretru.

Pro vyhodnocení výrazu jazyka Scheme stačí zavolat statickou metodu execute třídy `Context`, metodě lze mimojiné předat instanci třídy implementující rozhraní `SchemeCaller` s jedinou metodou `public Object execute(Interpreter) throws SchemeException`. Díky

této metodě se dostaneme k samotnému interpreteru, který například pomocí metody `Value eval(String)` vyhodnotí výraz obsažený v řetězci předaný jako argument. Jako výsledek vrátí vyhodnocený výraz reprezentovaný instancí třídy `Value`, tato třída poměrně dobře přepisuje metodu `toString()` takže výsledek vyhodnocení můžeme rovnou vytisknout na obrazovku, viz následující příklad:

```
// vytvoreni kontextu aplikace
AppContext ctx = new AppContext();

// ziskani cesty k halde
URL urlHeapPath = getClass().getResource("sisc.shp");

// prida haldu do~kontextu aplikace
ctx.addHeap(AppContext.openHeap(urlHeapPath));
Context.setDefaultAppContext(ctx);

// volani jazyka Scheme
Context.execute(new SchemeCaller() {
  @Override
  public Object execute(Interpreter interpreter)
            throws SchemeException {
    String expr = "((lambda (x y z) (cons x z)) ’a 11 ’b)";
    Value result = null;
    try {
      System.out.println("Evaulating expr: " + expr);
      // vyhodnoceni vyrazu
      result = interpreter.eval(expr);
      System.out.println("Result: " + result);
      return result;
    } catch (IOException e) {
      System.err.println("Error while evaulating expr: " + expr);
      System.err.println("Error message: " + e.getMessage());
      return null;
    }
  }
});
```

Metoda `eval(String)` může vyvolat výjimku – instanci třídy `IOException`, kterou je samozřejmě třeba zachytit. Výjimka může vzniknout například pokud je výraz reprezentovaný řetězcem špatně uzávorkovaný. Jestliže vše proběhne v pořádku, tak předchozí příklad vypíše na obrazovku:

```
Evaulating expr: ((lambda (x y z) (cons x z)) ’a 11 ’b)
Result: (a . b)
```

2.4. Používání jazyka Java z jazyka Scheme

V této části rozeberu jakým způsobem lze pracovat s jazykem Java z prostředí jazyka Scheme. Při této manipulaci projekt SISC používá generiku jazyka Java. Před zahájením pokusů o přístup k javovským objektům je nezbytné importovat modul `s2j` následovně:

```
(import s2j)
```

2.4.1. Převod elementů jazyka Scheme na objekty a zakladní typy jazyka Java

Modul `s2j` obsahuje procedury pro převod elementu jazyka Scheme na objekty a základní typy jazyka Java např.:

```
(->jint celé-číslo) => celé-číslo převedené na~typ int
(->jchar znak) => znak převeden na~typ char
(->jboolean pravdivostní-hodnota) =>
  pravdivostní-hodnota převedená na~typ boolean
(->jstring řětězec) => řětězec převeden na~typ String
```

Neuvedl jsem všechny procedury, názvy ostatních procedur jsou ve tvaru `->j"nazev-typu"`.

2.4.2. Vytvoření nové instance

Pokud chceme vytvořit instanci nějaké třídy musíme nejdříve získat objekt typu třída (v Jave potomek třídy `Class`). K tomuto účelu je procedura `java-class`, kterou lze použít například takto:

```
(java-class '|javax.swing.Jframe|) =>
  #<java java.lang.Class javax.swing.Jframe>
```

Procedura `java-class` požaduje jako svůj jediný argument symbol jednoznačně označující požadovanou třídu včetně označení balíčku, symbol musí být uvozen znakem `|` kvůli rozlišování velkých a malých písmen. Procedura vrací jako výsledek své aplikace instanci třídy `Class`. Nyní stačí zavolat konstruktor pomoci procedury `java-new`, která bere jako první argument objekt typu třída:

```
(java-new (java-class '|javax.swing.Jframe|)) =>
  #<java javax.swing.Jframe>
```

Procedura `java-new` vrací jako výsledek své aplikace instanci požadované třídy. Pokud chceme zavolat konstruktor, který vyžaduje nějaké argumenty, stačí je předat jako argumenty procedure `java-new`, argumenty musí byt opět převedeny na objekty z jazyka Java:

```
(java-new (java-class '|java.lang.Integer|) (->jint 11)) =>
  #<java java.lang.Integer 11>
```

2.4.3. Volání metody instance

Pro získání přístupu k metodě slouží procedura vyšších řádů `generic-java-method` se syntaxí:

```
(generic-java-method '|jmenoMetody|) => procedura
```

Pokud chceme například získat metodu `setVisible`, můžeme toho dosáhnout následovně:

```
(generic-java-method '|setVisible|) => #<procedure>
```

Procedura `generic-java-method` přijímá jako svůj argument symbol identifikující metodu (symbol musí byt opět uvozen znakem `|`), jako výsledek své aplikace vrací proceduru. Aplikace procedury (představující volání metody), požaduje jako první argument instanci objektu jazyka Java a požadovaný počet argumentů volané metody např.:

```
((generic-java-method '|setVisible|)
 (java-new (java-class '|javax.swing.JFrame|))
 (->jboolean #t)) =>
  #<jnull void>
```

2.4.4. Přístup k proměnným instance

Pro získání hodnoty proměnné instance slouží procedura `generic-java-field-accessor`, která vyžaduje jediný argument, symbol identifikující jméno proměnné instance. Tato procedura vrací jako výsledek své aplikace proceduru, která pro svůj jediný argument – objekt jazyka Java, vrací hodnotu požadované proměnné, například:

```
((generic-java-field-accessor '|MAX\_VALUE|)
 (java-new (java-class '|java.lang.Integer|)
           (->jint 11))) =>
  #<java int 2147483647>
```

3. Implementace aplikace

Grafické rozhraní (aplikaci) jsem vytvořil v programovacím jazyce Java, konkrétně Openjdk verze 6, s použitím projektu SISC, grafické knihovny Swing a jazyka Scheme. Jako vývojové prostředí jsem použil Eclipse Kepler a pro vývoj části aplikace v jazyce Scheme jsem použil samotnou aplikaci Sisc Scheme. Snažil jsem se klást maximální důraz na nezávislost rozhraní na projektu SISC, tak aby bylo možné používat aplikaci s minimálními úpravami, i s novou verzí projektu SISC.

Celá aplikace je logicky rozdělena do částí – balíčků, kde každý řeší specifickou část aplikace. V této sekci rozeberu způsob vytvoření jednotlivých balíčků postupně od nejnižší vrstvy.

3.1. Balíček sisc.core.resources

Tento balíček obsahuje soubor s haldou, a dva zdrojové soubory v jazce Scheme.

První z nich – `box-graph-module.scm` obsahuje modul `box-graph` s jedinou viditelnou procedurou `pair->graphstructure`, která umí převést jakýkoliv pár na seznam reprezentující graf páru v boxové notaci s ukazateli (zobrazován v inspektoru).

Druhý – `sisc-inspector` obsahuje modul se stejným jménem s procedurami `get-value-type` (pro získání jména typu předané hodnoty) a `get-value-structure` (pro získání seznamu popisující strukturu, výsledek této procedury je zobrazován v inspektoru).

3.2. Balíček sisc.core

Základní třídy obsažené v tomto balíčku jsou `SiscInterpreter` a `SiscCore`.

Třída `SiscCore` je základní třídou logické vrstvy aplikace nad Interpreterm projektu SISC. Třída `SiscCore` zapouzdřuje třídu `SiscInterpreter` a především spravuje paralelní přístup k interpreteru z jiných částí aplikace.

3.3. Balíček sisc.engine

Hlavní třídou balíčku je `SiscEngine`, který poskytuje zbytku aplikace metody pro vyhodnocování výrazů, pro hlášení dokončeného vyhodnocení, chyb atp.

3.4. Balíček sisc.inspector

Tento balíček obsahuje veškerou logiku aplikace pro inspekci a zobrazení struktur.

Při inspektování elementu je každý element navázán na vygenerovaný symbol jazyka Scheme. Inspektor během své činnosti vytváří výraz, po jehož vyhodnocení vzniknou konkrétní data, které lze zobrazit. Například pokud je zobrazena délka seznamu, tak pak je aktuální výraz `(length '|%%_IXoXPMBST|)`.

Jednou z nejzajímavejší tříd tohoto balíčku je abstraktní třída `SpecializedInspector`. Z této třídy dědí každý „specializovaný" inspektor, který umí manipulovat pouze se svým typem hodnot, například `PairSpecializedInspector` pracující pouze s páry. Třída `SpecializedInspector` má tyto metody:

```
public boolean isKeySettable(String key)
```

Metoda `isKeySettable` vrací `true` pokud lze hodnotu „klíče" elementu imperativně změnit. Například pro element typu pár vrací metoda `true` pouze pro klíče `car` a `cdr`, pro klíč `length` vrací `false`.

```
public Structure getDataToStructure(String type, String value,
        String[][] data)
```

Tato metoda vytvoří strukturu, kterou lze jednoduše zobrazit v inspektoru.

```
public StructurePathItem expand(String key)
```

Metoda slouží k expanzi „rozkliknutí" položky – jednoduše připojí na začátek aktuálního výrazu proceduru pro přechod o úroveň níž.

```
public abstract void set(String path, String key, String value)
        throws SetValueException
```

Metoda nastaví položku aktuálně zobrazené struktury nebo při neúspěchu vyvolá výjimku.

V tomto balíčku je i třída `Structure`, která je zobrazována v prezentační vrstvě aplikace (balíček gui.inspector).

Všechny třídy uvnitř tohoto balíčku zapouzdřuje třída `SiscInspector`.

3.5. Balíček sisc.syntax

Tento balíček obsahuje všechny třídy pro manipulaci se syntaxí jazyka Scheme. Pro některé zajímavé funkce aplikace, jako například automatické formátování zdrojového kódu, jsem musel vytvořit jednoduchý parser a ten představuje třída `SiscParser`.

3.6. Balíček gui.textpane

Uvnitř tohoto balíčku je především textové pole se schopností zvýrazňovat syntaxi ve zdrojovém kódu – třída `TextPane`. Dále balíček obsahuje jednotlivé prvky pro zobrazování polohy kurzoru, zvýrazňování závorek a nástroje pro editaci (formátování kódu, doplňování závorek, atp.).

3.7. Balíček gui.editor

Tento balíček obsahuje grafické prvky pro jednotlivé karty editoru a speciální nástroje používané pouze uvnitř editoru (např.: vyhodnocení výrazu do schránky).

Hlavní třídou balíčku je třída `SiscEditor`, který obsahuje karty editoru, nástojovou lištu, zobrazení polohy kurzoru atd.

3.8. Balíček gui.interaktor

V tomto balíčku je mimojiné třída `SiscInteraktor`, která představuje jeden z nejdůležitějších prvků aplikace. Třída si uchovává informace o jednotlivých výsledcích vyhodnocení a ty pak předává inspektoru k inspekci hodnot.

3.9. Balíček gui.inspector

Tento balíček slouží jako prezentační vrstva nad balíčkem sisc.inspector, třída `InspectorWindow` zobrazuje struktury reprezentované třídou `Structure`, které získává od instance třídy `SiscInspector`.

3.10. Balíček gui.graph

V tomto balíčku jsou třídy potřebné pro zobrazování párů v boxové notaci. Třída `Node` reprezentuje jeden box grafu a `RelationArrow` šipku mezi jednotlivými boxy. Všechny prvky jsou ukládány do kontejneru `GraphContainer`.

3.11. Balíček gui.debugger

Uvnitř tohoto balíčku jsou pouze dvě třídy `BreakpointDisplayer` a `SiscDebugger`. Třída `BreakpointDisplayer` slouží k zobrazování aktuálních breakpointů a třída `SiscDebugger` ke správě aktuálního breakpointu.

3.12. Balíček gui.settings

Hlavní třídy balíčku jsou třídy `Settings` (obsahuje informace o aktuálním nastavení a metody pro jeho uložení) a `SettingsWindow` (dialogové okno, pro nastavení klávesových zkratek).

3.13. Balíček gui.resources

Obsahem tohoto balíčku jsou veškeré ikony a třída `IconFactory` sloužící pro jejich snadný import do aplikace.

3.14. Balíček gui.application

V tomto balíčku je jediná třída – MainWindow, která slučuje všechny komponenty do jediného okna, navíc obsahuje i jednoduché menu.

3.15. Známé problémy

Aplikace poskytuje uživateli (programátorovi) možnost dynamicky měnit hodnoty v prostředí, samozřejmě i v globálním, nicméně neuváženou změnou může uživatel vyvolat u aplikace nepředvídatelné chování. Aplikace během své činnosti používá interpret a očekává že procedury, které používá, jsou navázány na svoje výchozí symboly.

Závěr

Téma bakalářské práce vycházelo z frustrace během používání aplikace Racket a nemožnosti nalézt alternativní grafické prostředí. Navíc možnost vytvořit vlastní grafické prostředí mě vždy lákala.

Výsledná aplikace Sisc Scheme umožnuje základní editaci, vyhodnocování výrazů jednoho po druhém, editovat jeden soubor ve více kartách, nastavit klávesové zkratky, automaticky formátovat kód, inspektovat struktury, destruktivně je měnit, zobrazovat je pomocí boxové notace, pracovat s breakpointy a procházet historii vyhodnocených výrazů.

Během implementace aplikace jsem narazil na problém s nemožností získání vzniklých prostředí po aplikaci procedury. Toto by bylo výborné rozšíření, které by bohužel znamenalo velký zásah do samotného interpreteru, kterému jsem se chtěl co nejvíc vyhnout.

Reference

[1] Kelsey R., Clinger W., Rees J. (editoři). *Revised^5 Report on the Algorithmic Language Scheme.* 1998.

[2] MIT/GNU Scheme. http://www.gnu.org/software/mit-scheme/. [Cit. 2014-06-05]

[3] Racket. http://racket-lang.org/. [Cit. 2014-06-05]

[4] Schemik. http://schemik.sourceforge.net/. [Cit. 2014-06-05]

[5] SISC. http://sisc-scheme.org/. [Cit. 2014-06-05]

[6] S. G. Miller, M. Radestock. *SISC for Seasoned Schemers.* 2006.

[7] TinyScheme. http://tinyscheme.sourceforge.net/home.html. [Cit. 2014-06-05]

[8] Wikipedia. http://en.wikipedia.org/wiki/Scheme_(programming_language). [Cit. 2014-06-05]

A. Příloha A: Uživatelská příručka

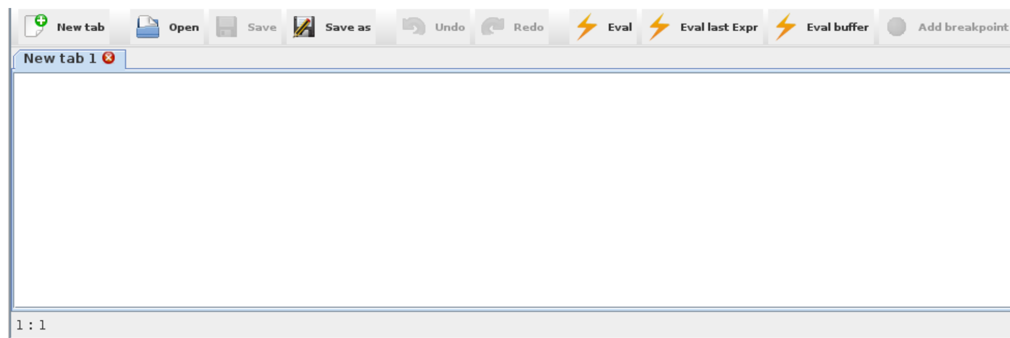
A.1. Spuštění aplikace

Aplikace vyžaduje pro své spuštění běhové prostředí Java verze 1.6 a vyšší. Aplikace je distribuovaná v podobě spustitelného archivu JAR, který obsahuje všechny potřebné zdroje pro své spuštění. Aplikaci lze spustit obyčejným kliknutím nebo pomocí příkazu:

```
java -jar SiscScheme.jar
```

A.2. Základní práce s editorem

Editor (obrázek 1.) se skládá z nástrojové lišty, textové oblasti a spodní lišty zobrazující aktuální polohu kurzoru.

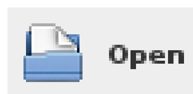


Obrázek 1. Editor.

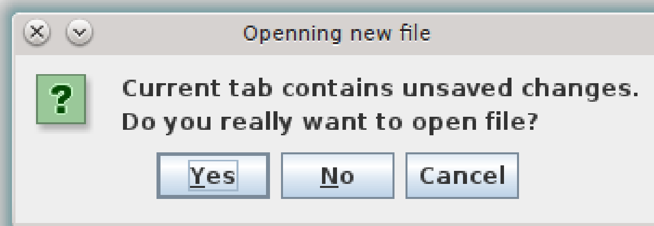
Pomocí tlačítka „New tab“ (obrázek 2.) lze vytvořit novou prázdnou kartu. Kartu lze zavřít pomocí křížku v jejím záhlaví.

Obrázek 2. Tlačítko pro vytvoření nové karty.

Otevření souboru do aktuální karty lze provést pomocí tlačítka „Open“ (obrázek 3.). Pokud jsou v aktuální kartě neuložené změny, tak aplikace zobrazí dotaz na potvrzení otevření souboru (obrázek 4.).



Obrázek 3. Tlačítko pro otevření souboru.



Obrázek 4. Dialog potvrzení otevření souboru.

Pro uložení současného obsahu karty slouží tlačítka „Save“ (obrázek 5.) a „Save as“ (obrázek 6.).

Obrázek 5. Tlačítko pro uložení obsahu karty.

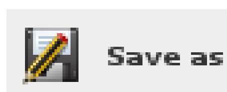
Editor obsahuje ve své nástrojové liště i tlačítka pro procházení historie posledních úprav. Pro vrácení poslední úpravy slouží tlačítko „Undo“ (obrázek 7.) a pro znovuprovedení úpravy tlačítko „Redo“ (obrázek 8.).

Editor i interaktor automaticky odsazují výraz na aktuálním řádku pomocí klávesy „tab“.

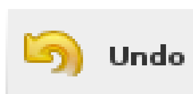
A.3. Vyhodnocování výrazu uvnitř editoru

Pro vyhodnocování výrazů v editoru jsou k dispozici tři tlačítka.

První tlačítko „Eval“ (obrázek 9.) vyhodnotí aktuálně označený výraz nebo pokud žádný označený není vyhodnotí výraz, ve kterém se nachází kurzor. Výsledek vyhodnocení zobrazí interaktor.



Obrázek 6. Tlačítko pro uložení souboru s novým jménem.



Obrázek 7. Tlačítko pro vrácení úprav.

Obrázek 8. Tlačítko pro znovuprovedení úprav.

Obrázek 9. Tlačítko pro vyhodnocení výrazu.

Druhé tlačítko „Eval last expr“ (obrázek 10.) vyhodnotí poslední výraz uvnitř aktuální karty.

Obrázek 10. Tlačítko pro vyhodnocení posledního výrazu.

Poslední tlačítko „Eval buffer“ (obrázek 11.) vyhodnotí celý obsah aktuální karty, podobně jako tlačítko „Run“ v aplikaci `Racket`.

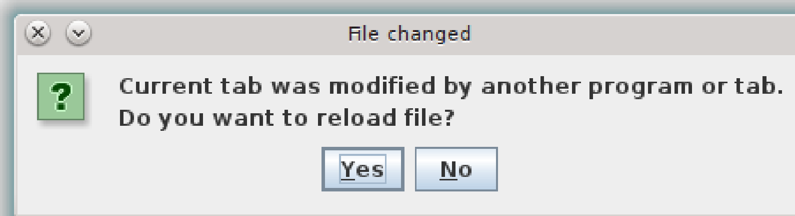
Obrázek 11. Tlačítko pro vyhodnocení celého obsahu karty.

Další možností jak vyhodnotit výraz je použít klávesovou zkratku pro vyhodnocení výrazu do schránky, ve výchozím nastavení je tato zkratka definovaná jako: `Ctrl + Alt + C`. Výsledek takového vyhodnocení není nikde zobrazen.

A.4. Otevření jednoho souboru ve více kartách

Jedním ze základních požadavků na aplikaci byla možnost otevřít jeden soubor ve více kartách. Toho lze dosáhnout otevřením dvou prázdných karet a do každé z nich nahrát stejný soubor.

Pokud uživatel provede změny v první kartě, změny uloží a poté zaktivuje druhou kartu, aplikace zobrazí dialog s dotazem zda uživatel chce změny nahrát do současné karty (obrázek 12.).



Obrázek 12. Dialog změny v otevřeném souboru.

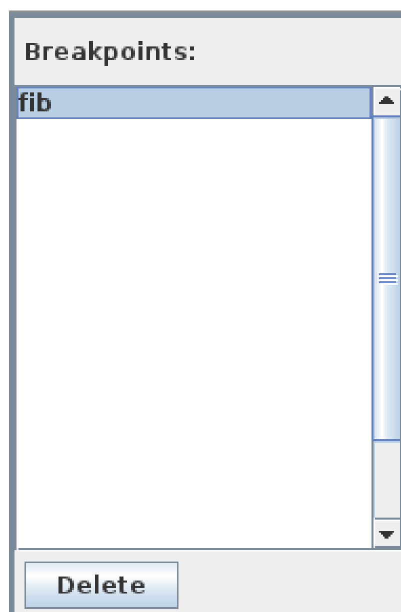
A.5. Práce s breakpointy

Breakpoint lze navázat na jakýkoliv symbol (na který je navázána procedura v globálního prostředí) v textové oblasti editoru.

Nejdříve je potřeba označit symbol na který má být breakpoint navázán a poté kliknout na tlačítko „Add breakpoint" (obrázek 13.).

Obrázek 13. Tlačítko pro přidání breakpointu.

Aktuální breakpointy jsou zobrazeny v samostatné části aplikace (obrázek 14.). V této části aplikace je lze odstraňovat pomocí tlačítka „Delete".

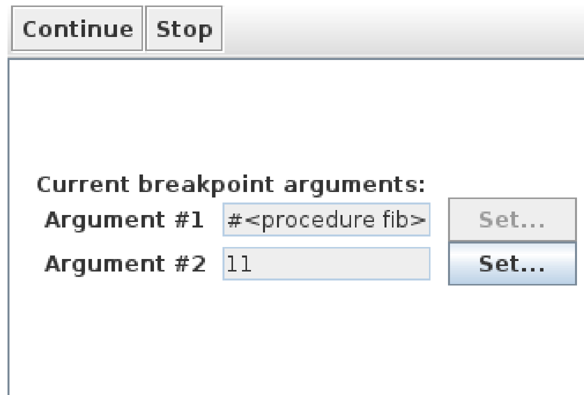


Obrázek 14. Seznam aktuálních breakpointů

Pokud je vyhodnocována procedura na kterou je navázán breakpoint, pak je výpočet pozastaven. Aktuální breakpoint včetně svých argumentů je zobrazen ve speciální části aplikace (obrázek 15.).

V této části aplikace lze postupně „krokovat" tlačítkem „Continue", přeskočit všechny následující breakpointy tlačítkem „Stop" nebo měnit aktuální hodnoty jednotlivých argumentů tlačítkem „Set…".

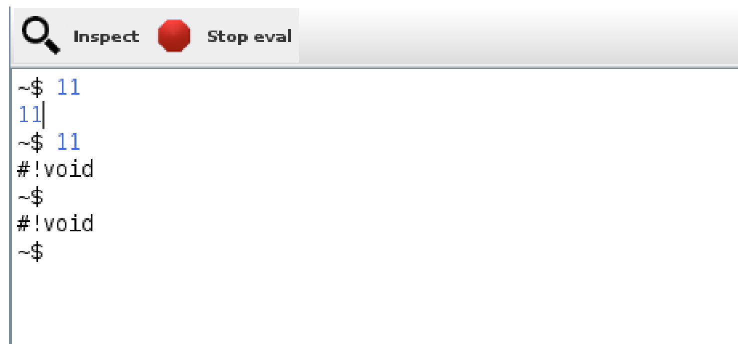
Na novou hodnotu argumentu breakpointu je uživatel dotázán dialogem, vložená hodnota je vždy vyhodnocena.



Obrázek 15. Zobrazení aktuálního breakpointu.

A.6. Práce s interaktorem

Interaktor (obrázek 16.) umožnuje kromě vyhodnocování výrazů, procházet historii vyhodnocovaných výrazů (v základním nastavení klávesovými zkratkami `Ctrl + šipka-nahoru` pro předchozí výraz a `Ctrl + šipka-dolů` pro následující výraz), ukončení aktuálního výpočtu a volání inspektoru.

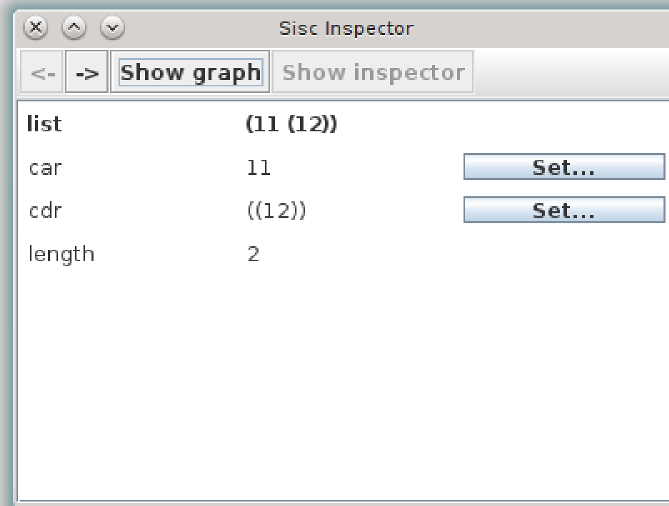


Obrázek 16. Interaktor.

A.7. Inspektovaní struktur

Téměř jakýkoliv výsledek zobrazený v interaktoru je možné zobrazit v inspektoru pomocí tlačítka „Inspect“ z interaktoru.

Inspektor (obrázek 17.) zobrazuje informace o aktuální struktuře nebo hodnotě, například pro strukturu z páru zobrazí hodnotu položky `car`, `cdr` a délku páru.



Obrázek 17. Inspektor.

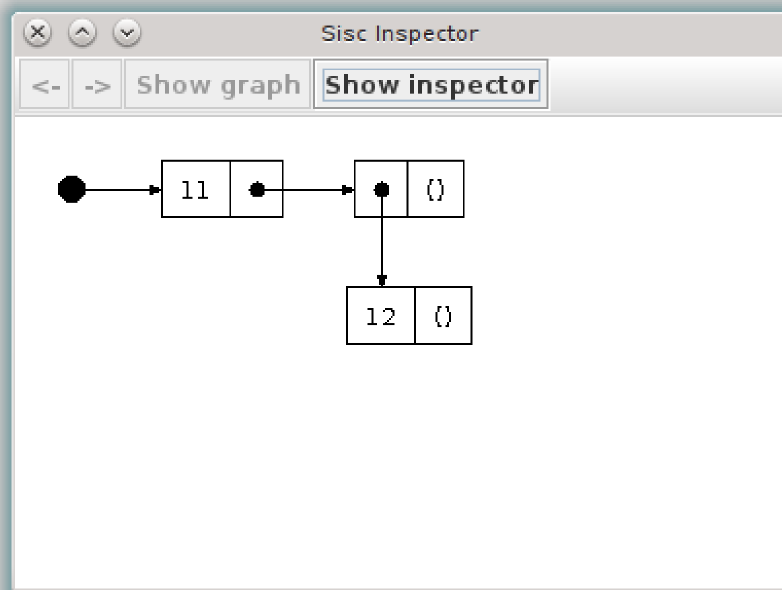
Zobrazené položky lze procházet dvoj-kliknutím. Inspektor si uchovává historii procházení, tu lze ovládat tlačítkami v horním menu (šipka vpravo a šipka vlevo).

Většinu hodnot položek struktur lze dynamicky změnit tlačítkem „Set. . .“. Po kliknutí na tlačítko „Set. . .“ je vyvolán dialog, který se dotáže uživatele na novou hodnotu položky.

A.8. Grafické zobrazení struktur

V horním menu interaktoru je prostřednictvím tlačítka „Show graph“ možné zobrazit strukturu z párů pomocí boxové notace s ukazateli (obrázek 18.). Při zobrazení boxové notace jsou jednotlivé boxy naskupené na jedné hromadě.

S jednotlivými boxy lze manipulovat pomocí myši metodou drag and drop.



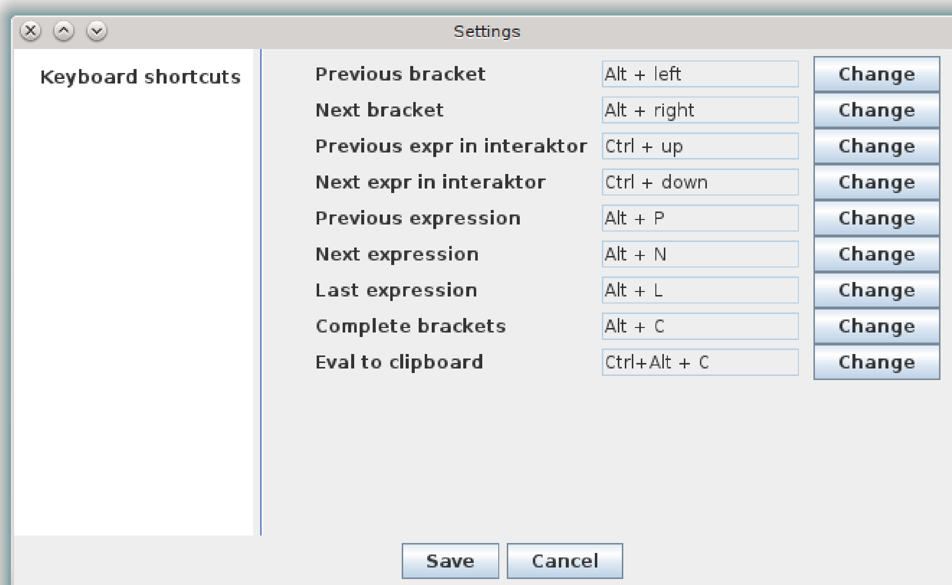
Obrázek 18. Struktura zobrazená pomocí boxové notace s ukazateli.

A.9. Nastavování klávesových zkratek

Funkce pro změnu klávesových zkratek je dostupná z horního menu „Edit“ prostřednictvím položky „Settings“.

Klávesové zkratky lze nastavit ve speciální dialogu (obrázek 19.). V tomto dialogu, po kliknutí na tlačítko „Change“, stačí stisknout požadovanou kombinaci kláves.

Nastavení klávesových zkratek se ukládá do domovského adresáře `.siscScheme/` v Unixových systémech nebo do `Application Data\siscScheme\` v operačních systémech od firmy Microsoft Windows.



Obrázek 19. Dialog pro nastavení klávesových zkratek.

B. Obsah přiloženého CD

`bin/`
: Zkompilovaná verze aplikace Sisc Scheme v podobě spustitelného jar souboru.

`doc/`
: Bakalářská práce ve formátu PDF a ZIP soubor se zdrojovými soubory práce a obrázky.

`src/`
: Kompletní zdrojové soubory aplikace včetně souborů nutných pro bezproblémové spuštění aplikace.

`readme.txt`
: Instrukce pro spuštění aplikace.