MASARYKOVA UNIVERZITA • PŘÍRODOVĚDECKÁ FAKULTA

Bakalářská práce

SYSTÉM POČÍTAČOVÉ ALGEBRY AXIOM

Brno, květen 2007 Pavel Hellebrand

Poděkování

Chtěl bych poděkovat RNDr. Romanu Plchovi, Ph.D. za vedení bakalářské práce, cenné rady a připomínky při zpracování daného tématu.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval sám, pouze za pomoci RNDr. Romana Plcha, Ph.D. a uvedené literatury.

V Brně dne 18. května Pavel Hellebrand

Obsah

Úvod

Axiom patří do skupiny systémů počítačové algebry, do které spadají například i komerční programy jako Maple nebo Mathematica.

Počátek vývoje systému Axiom sahá až do 70. let minulého století, kdy byl vyvíjen pod názvem Scratchpad a sloužil pouze k vědeckým účelům. V devadesátých letech dostal název Axiom a byl distribuován společně s rozsáhlou uživatelskou příručkou komerční cestou. Na konci devadesátých let bylo rozhodnuto, že systém Axiom bude nadále šířen jako program s otevřeným zdrojovým kódem a to zdarma. Nyní je dostupný k bezplatnému stažení z internetových stránek `http://wiki.axiom-developer.org/FrontPage` a to jak pro systém Windows, tak pro systém Linux (pro který je vyvíjen přednostně). Na výše uvedených stránkách je rovněž k dispozici rozšířená verze manuálu (s kterým byl původně šířen), nazvaná Axiom book 2. Protože je program šířen s otevřeným zdrojovým kódem, tak se na jeho dalším vývoji může podílet každý s dostatečnými znalostmi a jako takový je neustále vylepšován.

Systém Axiom skýtá obrovské možnosti (vždyť manuál má v rozsahu přes tisíc stránek) od jednoduchých numerických výpočtů až po složité konstrukce obecné algebry. Každý uživatel si jistě najde svůj okruh funkcí, které ke své praxi potřebuje. Vybraný okruh potřebných funkcí si lze osvojit velice rychle, neboť většina funkcí se používá zcela intuitivně a někdy postačí pouhá znalost angličtiny (viz funkce integrate, solve, ...).

Instalace

Na domovských stránkách programu nejprve vybereme mutaci systému Axiom podle toho, pod jakým operačním systémem chceme program používat. Dostupné mutace jsou pro operační systémy Windows a pro nejběžnější vydání systému Linux jako je Debian, Fedora a některé další. Po stažení potřebných souborů (u každé varianty je uveden jejich seznam s příslušnými odkazy pro stažení) postupujeme dále podle pokynů uvedených přímo u těchto souborů.

Instalace pod Windows

Při instalaci pod systémem Windows stačí stáhnout pouze jeden instalační soubor, který nás po spuštění sám provede celým procesem instalace. Instalace je realizována běžným způsobem tak, jak je tomu u naprosté většiny programů pod tímto operačním systémem. U systému Windows nelze nyní používat grafické rozhraní a systém funguje pouze v textovém režimu. V současné době se na grafickém prostředí pro Windows pracuje. Pokud chceme grafické prostředí ve Windows používat již nyní, tak jediná cesta je pomocí emulátoru linuxového prostředí. Tento emulátor je i s návodem na jeho instalaci dostupný rovněž na domovských stránkách systému Axiom.

Instalace na systému Linux

Postup zde uvedený by měl fungovat na většině distribucí operačního systému Linux, hlavně na verzích Fedora a Debian. Na stránkách `http://wiki.axiom-developer.org/AxiomBinaries` je však podrobně popsán postup instalace pro většinu mutací systému.

Na výše uvedených stránkách stáhneme potřebný soubor, či soubory (dle verze). Soubory rozbalíme do domovského adresáře.

```
1) cd /home
2) tar -zxf axiom-aldor-20060621.tgz
```

Jméno souboru za příkazem `tar` se může v jednotlivých verzích lišit. Samotnou instalaci provedeme posloupností příkazů

```
3) export AXIOM=/home/axiom/mnt/linux
4) export PATH=$AXIOM/bin:$PATH
```

A spustíme příkazem

```
5) axiom
```

Základní funkce systému Axiom

1.1 Spouštění systému Axiom a popis pracovního prostředí

Pro potřeby této práce budeme systém axiom spouštět z terminálu systému Linux, kde také celý systém běží. Pokud chceme terminál používat k dalším příkazům, je vhodné otevřít si před spuštěním Axiomu terminál nový. Systém se spouští příkazem **axiom**[1]. Po zadání zmíněného příkazu se v terminálu objeví tento nebo podobný text:

```
                AXIOM Computer Algebra System
              Version: Axiom 3.9 (September 2005)
        Timestamp: Thursday December 22, 2005 at 15:05:42
------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------

(1) ->
```

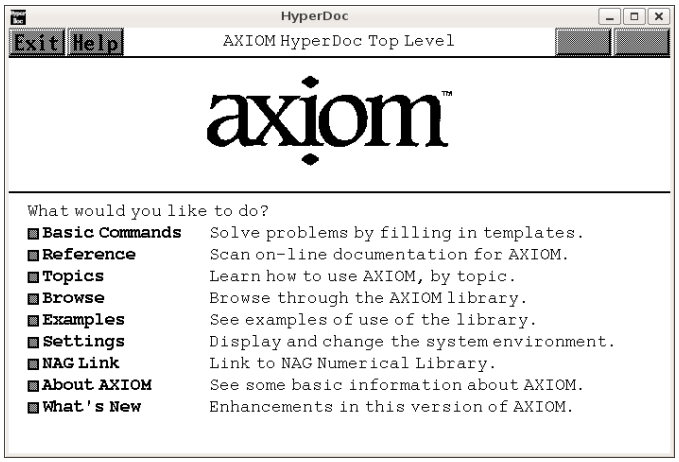
V první části tohoto textu nalezneme informace o programu a jeho verzi. V druhé části je seznam užitečných příkazů a část třetí je místo pro zadávání příkazů. Pokud spouštíme Axiom v grafickém prostředí Linuxu, tak se také otevře ještě jedno samostatné okno, které se nazývá Hyperdoc a slouží jako systém nápovědy. V tomto okně je navíc možné vyhledané příkazy přímo spouštět (obrázek 1.1). Jak vypadá výpočet v systému Axiom si ukážeme na zcela elementárním výpočtu, jako je $1 + 2$ a popíšeme si jednotlivé části výstupu.

[1]Spouštění pod systémem Windows se provádí buď stejným způsobem přes příkazový řádek, nebo přes příslušný spouštěcí soubor

```
(1) -> 1+2
                                                    Type: PositiveInteger
```

(1) 3

Text `(1) ->` označuje začátek příkazového řádku, do kterého můžeme zadávat příkazy. Číslo v závorce je číslo prováděného kroku, které může posloužit k odkazování se na výsledky předchozích výpočtů. Číslo kroku je také zobrazeno před výsledkem, aby bylo zřejmé, ke kterému kroku výsledek patří. Toto může být velmi důležité, protože mezi příkazem a výsledkem může být i několik stránek textu o prováděných operacích. Poslední řádek obsahuje informaci o datovém typu výsledku. Typ výsledku může pomoci vypátrat chyby v zadání, pokud nedostaneme očekávaný výsledek.



Obrázek 1.1: Základní menu Hyperdoc

1.2 Systémové příkazy

Systémové příkazy jsou používány k ovládání některých funkcí Axiomu, jako je nápověda, informace o příkazech, změna formy výstupu, nastavení domovského adresáře atd. Některé příkazy jsou standardně potlačeny z důvodu prevence před zásahem méně zkušeného uživatele a následného poškození funkčnosti systému. Pro práci se systémovými příkazy jsou definovány tři stupně pokročilosti: *Interpreter* pro běžného uživatele (jsou přístupné pouze

příkazy neohrožující stabilitu systému), *Compiler* a *Development* (zpřístupněny jsou všechny systémové příkazy) určeny zejména pro programátory, kteří se chtějí zabývat modifikacemi celého systému. Základní uživatelská úroveň je nastavena na úroveň *Interpreter*. Mezi jednotlivými úrovněmi lze přepínat pomocí funkce **)set userlevel** následované jedním z názvů úrovně. Příklady systémových příkazů, které zde uvedeme, budou nejzákladnější a všechny přístupné z úrovně *Interpreter*.

Všechny systémové příkazy jsou uvedeny znakem „) “. Za tímto znakem následuje příkaz a za ním argument oddělený od příkazu mezerou (některé funkce nemusí mít argument žádný), který se narozdíl od výpočetních příkazů nepíše do závorky. Dále může systémový příkaz obsahovat volitelné argumenty oddělené mezerami nebo uzavřené do jednoduchých závorek, podle syntaxe jednotlivých příkazů. Zde následuje seznam některých systémových příkazů:

- **)cd *adresář*** pro nastavení pracovního adresáře
- **)close** ukončí poslední uživatelem prováděnou operaci. Pokud je tímto ukončena poslední běžící operace, dojde k ukončení celého systému Axiom. Uživatel je předem upozorněn.
- **)clear all** smaže pracovní plochu a nastaví počitadlo kroků na jedničku.
- **)clear value all** smaže všechny uživatelem deklarované proměnné. Pro smazání konkrétních proměnných se místo slova **all** napíšou jednotlivé jména proměnných oddělené mezerami. Počitadlo kroků zůstane na předchozí hodnotě.
- **)display names** vypíše seznam jmen všech systémem a uživatelem definovaných objektů.

```
    -> )display names
Names of User-Defined Objects in the Workspace:

%    x    y

Names of System-Defined Objects in the Workspace:

%e                %i          %infinity
%minusInfinity    %pi         %plusInfinity
SF
```

- **)display properties all** vypíše seznam jmen všech definovaných objektů spolu s informacemi o datových typech a přiřazených hodnotách.
- **)read *jmeno_souboru*** Načte obsah vstupního souboru (viz oddíl 1.8.1). Soubor musí být uložen v pracovním adresáři.
- **)quit** vypne Axiom a vrátí nás do prostředí, ve kterém jsme ho spouštěli. Před vypnutím budeme vyzváni k potvrzení volby.

1.3 Systém nápovědy

Jako každý jiný systém počítačové algebry má i Axiom zabudován rozsáhlý systém nápovědy pro všechny funkce v systému obsažené i k fungování systému samotného. Možnosti pro vyhledání potřebné informace máme v podstatě dvě:

- okno Hyperdoc
- příkazový řádek systému Axiom.

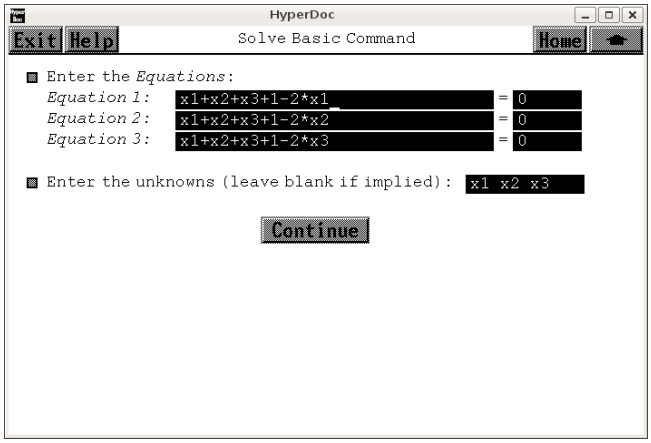
1.3.1 Hyperdoc

První možností pro vyhledání potřebné informace je použití okna Hyperdoc. Hyperdoc je interaktivní rozhraní, ve kterém se postupným klikáním na témata, o která se zajímáme, dostaneme k hledané informaci. Síla této možnosti spočívá v tom, že k vyhledání příkazu postačí vědět pouze, jaký problém chceme řešit. Navíc příkaz z tohoto rozhraní jedním kliknutím spustíme i se specifikací našich hodnot. Pokud se dostaneme do podvětve, kde danou informaci nedostaneme, tak pomocí zabudovaných tlačítek se kdykoliv vrátíme o úroveň nahoru nebo na úvodní stránku okna Hyperdoc (obrázek 1.1).

Nejlépe si ukážeme tuto možnost na konkrétním příkladě. Chceme vyhledat funkci pro řešení soustavy lineárních rovnic a rovnou danou soustavu vyřešit. Najdeme ji touto posloupností odkazů:

Basic Commands → Solve → A System Of Linear Equations → Directly As Equation.

Následně budeme vyzváni k zadání počtu rovnic a poté k jejich přesnému zadání (obrázek 1.2). Po zadání rovnic Hyperdoc vypíše přesný tvar příkazu, který danou soustavu vyřeší a po zmáčknutí tlačítka *Do It* příkaz spustí na příkazovém řádku Axiomu.



Obrázek 1.2: Okno pro zadávání soustavy rovnic

1.3.2 Nápověda přes příkazový řádek

Narozdíl od okna Hyperdoc musíme vědět jaký příkaz chceme použít a přes nápovědu vyhledáme tvar, ve kterém je nutno příkaz zadávat. Druhou možností je nechat si vypsat seznam všech příkazů ze stejné rodiny, např. všechny systémové příkazy, příkazy pro lineární algebru atd. a ze seznamu vybrat vhodnou funkci.

Pro nápovědu pro konkrétní funkci se používá funkce **)help *funkce***. Pokud chceme vypsat seznam funkcí (např. seznam všech systémových příkazů, které jsou přístupné ze zapnuté uživatelské úrovně), tak použijeme příkazu **)what commands**. Další parametry, které lze použít místo parametru commands a vypsat tak jiný seznam funkcí, získáme samotným zadáním příkazu **)what**.

Pokud známe jen část hledaného příkazu tak funkce **)what op *hledaný_řetězec*** vypíše seznam všech příkazů, které obsahují hledaný řetězec v jejich názvu.

1.4 Provádění výpočtu bez výstupu na obrazovku

V některých případech, zvláště pokud prováděný výpočet je pouze mezikrokem nějakého složitějšího výpočtu, není potřeba, aby Axiom vypisoval výstup příkazu na obrazovku. Pokud tedy nechceme výsledek vypsat, pak zadání příkazu ukončíme středníkem. Tímto způsobem lze napsat na jediný řádek i

více příkazů, přičemž jednotlivé příkazy oddělíme středníkem. Axiom vypíše pouze výsledek posledního příkazu, pokud také není ukončen středníkem.

```
-> 2+3*4;
                                        Type: PositiveInteger
```

1.5 Přístup k předchozím výsledkům

Makro % reprezentuje výsledek předchozí operace. Makro %% následované celočíselným argumentem reprezentuje výsledek kroku podle čísla v závorce. Pokud tedy chceme v nějakém výpočtu použít výsledek, který jsme dostali v pátém kroku, použijeme makro **%%(5)**. Pokud do argumentu makra %% napíšeme záporné číslo, tak tím zpřístupníme výsledek operace o tolik kroků dříve, kolik je absolutní hodnota tohoto čísla. Makro **%%(-1)** má stejný výsledek jako makro %.

1.6 Dělení výrazů přes více řádků

I když Axiom akceptuje výrazy, které jsou výrazně delší než je jeden řádek (stačí pouze psát bez stisku klávesy **ENTER**), je často lepší rozdělit výraz do více řádků tak, aby byl výraz lépe čitelný. K provedení tohoto stačí umístit znak „podtržítko“ před místo, ve kterém chceme dělení provést a následně stisknout klávesu **ENTER**. Se zapisováním zbytku výrazu se pokračuje na dalším řádku a může být uvozen libovolným počtem mezer pro zlepšení čitelnosti. Axiom mezery následující po znaku podtržítko ignoruje. Například:

```
-> 2_
   +_
   2

                                   4

                                        Type: PositiveInteger
-> Totojevelmidlouhe_
   jmenopromenne

          Totojevelmidlouhejmenopromenne
           Type: Variable Totojevelmidlouhejmenopromenne
```

1.7 Výstup do TEXu

Axiom kromě tradičního výstupu umožňuje nastavit i jiné formáty výstupu. Jedním z nich je výstup v kódu podporovaném sázecím systémem TEX. Formát výstupu lze nastavit tak, aby se vypisoval už přímo ve zdrojovém kódu zpracovatelném systémem TEX. Takto nám odpadá práce s přepisováním výsledků výpočtů. Výstupní kód stačí pouze běžným způsobem zkopírovat do cílového dokumentu.

Funkce, pomocí které změníme formát výstupu na zdrojový formát pro TEX, je **)set output tex on**. Pokud se chceme vrátit zpět ke standardnímu formátu výstupu použijeme funkci **)set output tex off**. S takto nastaveným formátem výstupu můžeme pracovat obvyklým způsobem, pouze výsledek se nejdřív vypíše v standardním tvaru a hned pod ním je stejný výsledek, ale ve zdrojovém kódu sázecího systému TEX.

```
(1) -> e^(cos(x^3))+sqrt(x)
```

(1) $$e^{\cos\left(x^3\right)}+\sqrt{x}$$

```
$$
{e \sp {\cos
\left(
{{x \sp 3}}
\right)}}+{\sqrt
{x}}
\leqno(1)
$$
```

```
                                    Type: Expression Integer
```

1.8 Vstupní a výstupní soubory Axiomu

Od každého výpočetního systému se očekává, že si budeme moci výsledky početních operací v nějaké formě uchovat a načíst k dalšímu zpracovávání. Stejně je tomu tak u Axiomu. Axiom dokáže pracovat jak s vlastními soubory, které jsou čitelné jen Axiomem, tak se soubory v textovém formátu, které můžeme nejen sami prohlížet, ale i dále zpracovávat dalšími programy.

Soubory dělíme na vstupní a výstupní. Soubory v textovém formátu, můžeme upravovat a vytvářet v textových editorech nezávisle na Axiomu, nebo v Axiomu pomocí příkazu **)edit**. Příkaz automaticky soubor otevře v textovém editoru, který má nastavený pro tuto operaci. V systému Linux

je to editor **vi**. Pokud tento editor není v systému nainstalován, pokusí se Axiom otevřít nějaký jiný ze seznamu podporovaných editorů. Pokud chceme nastavit spouštění jiného textového editoru, tak v příkazovém řádku Linuxu napíšeme příkaz **export EDITOR=jmeno_editoru**.

Dále Axiom může vytvářet soubory datového typu File, které vytváříme přímo v Axiomu pomocí příkazů **open!**, **close**, **writeln!**, **readln!** a jiných, způsobem podobným např. práci se soubory v programovacím jazyce Pascal. Tento způsob práce necháme pro samostatné nastudování a zde se budeme věnovat pouze základním možnostem.

1.8.1 Vstupní soubor

Jako vstupní soubor Axiomu používáme textové soubory s koncovkou **.input**. Tyto soubory můžeme vytvořit v libovolném textovém editoru, nebo je můžeme vytvářet přímo v Axiomu.

Vstupní soubor Axiomu může obsahovat jakékoliv příkazy, které jsou dostupné v samotném systému. Vstupní soubor využijeme především pro načítání proměnných, které chceme opakovaně využívat, složitějších datových struktur (např. matice) nebo vlastních funkcí.

Vstupní soubor musí být uložen v adresáři, ve kterém máme Axiom nainstalovaný, nebo v adresáři, který máme nastavený jako pracovní. Pracovní adresář může být ten, ve kterém Axiom spustíme nebo jiný nastavený pomocí funkce **)cd**.

Načtení vstupního souboru provedeme příkazem **)read *jmeno_souboru***. Pokud neuvedeme jméno souboru, tak se načte soubor, který jsme načetli jako poslední, nebo naposledy upravovali příkazem **)edit**.

Speciálním vstupním souborem je soubor **axiom.input**. Tento soubor je prvním souborem, který se Axiom pokusí načíst po spuštění. Obsahem souboru jsou systémové příkazy, které určují některé chování pracovního prostředí jako je formát výstupu atd. Editací tohoto souboru můžeme prostředí Axiomu upravit pro svoji potřebu. Takto upravený soubor musí být uložen v pracovním adresáři. Pokud v pracovním adresáři není, tak dojde k načtení původního souboru, který je uložen v adresáři **src/input**.

1.8.2 Výstupní soubory

Pro tvorbu výstupních souborů máme několik možností. Zde zmíníme jenom základní dvě.

Příkazem **)spool *soubor.out*** zahájíme záznam veškeré práce s Axiomem do stejnojmenného textového souboru v pracovním adresáři (pokud neexistuje bude vytvořen). Do tohoto souboru se zaznamenávají jak naše příkazy,

tak všechny textové výstupy Axiomu. Tento soubor pak můžeme otevřít v libovolném textovém editoru, a dále s ním pracovat. Ukončení záznamu práce provedeme zadáním příkazu **)spool** beze jména souboru.

Dalším formátem výstupu je soubor pro ukládání historie práce s Axiomem. Tento soubor má koncovku **.axh** a Axiom jej ukládá standardně do pracovního adresáře. V tomto adresáři jsou ukládány informace o poslední podobě pracovního prostředí. Tj. všechny provedené operace i s číslem kroku a výsledkem. Díky historii můžeme používat příkazy pro odkazování se na předchozí výsledky **%** a **%%** (viz sekce 1.5). Tento soubor není klasickým textovým souborem a může být používán pouze Axiomem. Soubor se vždy smaže s vypnutím Axiomu.

Pokud chceme historii uložit pro budoucí použití, tak použijeme příkaz **)history)save *jmeno_souboru***. Pro načtení takto uloženého souboru historie použijeme příkaz **)history)restore *jmeno_souboru***. Tento postup je vhodný zejména pokud jsme nuceni ukončit práci s Axiomem uprostřed nedokončené práce. Příkaz **)history)write *soubor*** z historie vygeneruje vstupní soubor **soubor.input**.

Jednoduché numerické výpočty

2.1 Základní operace

Jako příklad numerického výpočtu v systému Axiom, nám poslouží výpočet kosinu 2,45 (v radiánech). Výpočet tohoto příkladu vypadá takto:

```
-> cos(2.45)
```

$$-0.7702312540473073417$$

```
                                                    Type: Float
```

Aritmetické operace (jako sčítání, odčítání a násobení) se chovají, jak očekáváme:

```
-> 7.34*5.2376
```

$$38.443984$$

```
                                                    Type: Float
```

```
-> 7.34/5.2376
```

$$1.4014052237666106614$$

```
                                                    Type: Float
```

Ale celočíselné dělení není tak zřejmé. Například, při zadání

```
-> 4/6
```

$$\frac{2}{3}$$

```
                                                    Type: Fraction Integer
```

získáme výsledek ve tvaru zlomku, zatímco v dalším případě je typ stále `Fraction Integer`, ale zobrazení je jiné. Je to dáno tím, že funkce, použitá k tomuto výpočtu, je naprogramována tak, aby výsledek zobrazovala v nejlépe čitelném tvaru.

```
-> 4/2
```

$$2$$

```
                                                    Type: Fraction Integer
```

Výsledek je uložen v paměti jako zlomek 2/1, ale je zobrazen jako celé číslo 2. Tento zlomek může být převeden na typ `Integer` bez ztráty informace, ale systém Axiom to nedělá automaticky.

2.2 Převod mezi jednotlivými typy výsledků

K získání hodnoty zlomku ve formě desetinného čísla (typ `Float`) je třeba provést konverzi použitím operátoru „**::**“, jak je ukázáno v následujícím příkladě.

```
-> (25/3)::Float
```

$$8.3333333333333333$$

```
                                                    Type: Float
```

I když Axiom dokáže převést výsledek zpátky do tvaru zlomku, zlomek nemusí být stejný, protože může dojít k chybě způsobené zaokrouhlováním.

```
-> %::Fraction Integer
```

$$\frac{25}{3}$$

```
                                                    Type: Fraction Integer
```

Tento výsledek je sice bez chyby, ale u jiných příkladů tak být nemusí.

Přestože je Axiom schopný pracovat s desetinnými čísly s velkou přesností, je třeba mít na paměti, že práce s těmito čísly není exaktní. Pro maximální přesnost by tedy měl uživatel nejdřív Axiom použít ke všem symbolickým výpočtům a až nakonec provést numerické vyjádření výsledku.

Operátor „**::**“ se používá k převodu hodnot z jednoho typu do druhého. Tento typ převodu nemůže být používán kdykoli. Například číslo 4,3 nemůže být převedeno na typ `Integer`, ale číslo 4,0 ano. Pro převod z typu `Float` do typu `Integer` je třeba, aby desetinná část reálného čísla byla nulová.

Převod mezi typy `Float` a `Integer` může být také realizován pomocí funkcí **round** a **truncate**. První z těchto funkcí převede desetinné číslo na nejbližší celé číslo, zatímco funkce **truncate** pouze odstraní část za desetinnou čárkou. Obě tyto funkce vrací výsledek typu `Float`, který může být pomocí operátoru „**::**“ převeden na typ `Integer`. Pro získání desetinné části se používá funkce **fractionPart**. Znaménko výsledku závisí na znaménku celé části čísla.

```
-> round(5.77653)
```

$$6.0$$

```
                                                    Type: Float
```

```
-> round(-5.77653)
```

$$-6.0$$

```
                                                    Type: Float
```

```
-> truncate(5.77653)
```

$$5.0$$

```
                                                    Type: Float
```

```
-> fractionPart(-5.77653)
```

$$-0.77653$$

```
                                                    Type: Float
```

2.3 Komplexní čísla

Pro mnohé vědecké výpočty nejsou reálná čísla dostačující, a proto Axiom podporuje i výpočty s komplexními čísly. Operace s komplexními čísly jsou zde zpracovány velice intuitivním způsobem pomocí makra "`%i`", které reprezentuje druhou odmocninu čísla -1. Výpočty s komplexními čísly jsou zapisovány stejně jako ostatní výpočty.

```
(1) -> (3/4+%i)**2
```

$$(1) \qquad \frac{-7}{16} + \frac{3}{2}\, i$$

```
                                   Type: Complex Fraction Integer
```

Reálná a imaginární část komplexního čísla může být získána pomocí funkce **real** a **imag**. Číslo komplexně sdružené se vypočítá pomocí funkce **conjugate**.

```
(2) -> imag(%)
```

$$(2) \qquad \frac{3}{2}$$

```
                                      Type: Fraction Integer
```

```
(3) -> conjugate(%%(1))
```

$$(3) \qquad \frac{-7}{16} - \frac{3}{2}\, i$$

```
                                   Type: Complex Fraction Integer
```

2.4 Datové struktury v Axiomu

V Axiomu je nepřeberné množství datových struktur. V této části se budeme věnovat pouze strukturám typu Seznam(List), a zmíníme se i o typu Stream, Set a Record.

Nejpoužívanější datovou strukturou je typ Seznam. Slouží k ukládání objektů stejného datového typu (např. Integer, Real, String, Equation, ...). U této datové struktury záleží na pořadí prvků a může obsahovat i duplikáty hodnot. Počet prvků seznamu musí být konečný.

Seznam se v Axiomu vytváří zápisem jednotlivých prvků do hranatých závorek, přičemž jednotlivé prvky musejí být odděleny čárkou. Následující příklad ukazuje vytvoření seznamu o prvcích, 1, 2 a $\frac{3}{2}$.

```
(1) -> u:=[1,2,3/2]
```

(1) $$\left[1,\ 2,\ \frac{3}{2}\right]$$

```
                                    Type: List Fraction Integer
```

Pokud tvoříme seznam, kde jednotlivé prvky jsou svázány nějakým pravidlem , můžeme použít tohoto pravidla i při tvorbě tohoto seznamu. Například tvorba seznamu prvočísel od jedné do desíti použijeme příkazu

```
(2) -> v:=[i for i in 1..10 |prime? i]
```

(2) $$[2,\ 3,\ 5,\ 7]$$

```
                                    Type: List PositiveInteger
```

K jednotlivým prvkům seznamu přistupujeme zadáním jména seznamu a číslem pozice prvku v závorce, nebo ve formátu **seznam.pozice**. Oba následující příkazy přiřadí proměnné x druhý prvek seznamu u.

```
(3) -> x:=u(2);
                                    Type: Fraction Integer
(4) -> x:=u.2
```

(4) $$2$$

```
                                    Type: Fraction Integer
```

Příkaz **#(u)** vrací počet prvků v seznamu u. Příkaz **removeDuplicates** odstraní ze seznamu všechny duplikátní hodnoty a zanechá pouze první výskyty jednotlivých prvků. Příkazem **member?** otestujeme, zda se zadaný prvek vyskytuje v seznamu. Pro obrácení pořadí prvků v seznamu použijeme příkazu **reverse**.

```
(5) -> member?(3,v)
```

(5) $true$

Type: Boolean

Jednotlivé prvky seznamu můžeme libovolně měnit. Pokud chceme změnit druhý prvek seznamu u na hodnotu 99 použijeme příkaz **u(2):=99**. Toto přiřazení je destruktivní, takže pokud používáme seznam u například v dalším seznamu k, tak se tento prvek změní i v tomto seznamu. Tento seznam k obsahující u vytvoříme třeba těmito dvěmi způsoby, z nichž ten druhý je použitím příkazu na spojení dvou seznamů. Tento příkaz můžeme využít i pro přidávání nových prvků do již existujícího seznamu.

```
(6) -> k:=append(u,v)
```

(6) $\left[1,\ 2,\ \frac{3}{2},\ 2,\ 3,\ 5,\ 7\right]$

Type: List Fraction Integer

```
(7) -> u.2:=99;
```

Type: Fraction Integer

```
(8) -> k
```

(8) $\left[1,\ 99,\ \frac{3}{2},\ 2,\ 3,\ 5,\ 7\right]$

Type: List Fraction Integer

Stream je datová struktura, která, narozdíl od seznamu, může obsahovat nekonečný počet prvků. Operace, které můžeme použít pro seznam, můžeme použít i pro stream. Axiom standardně zobrazuje prvních deset prvků této struktury, ale pracovat můžeme s libovolným prvkem pomocí výše uvedených způsobů přístupu k jednotlivým prvkům. Pro změnu počtu zobrazovaných prvků slouží příkaz **)set streams calculate**. Následující příkaz vytvoří proměnou s typu stream, jejíž prvky budou všechny lichá kladná čísla.

```
    -> s:=[i for i in 1.. | odd? i]
```

$$[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, \ldots]$$

```
                                        Type: Stream PositiveInteger
```

Další datovou strukturou je Set (množina). Rozdíl mezi strukturami množina a seznam spočívá v tom, že množina nemůže obsahovat duplicitní hodnoty a nezáleží v ní na pozici prvku. Počet prvků v množině musí být vždy konečný a neexistuje žádná možnost vytvořit nekonečnou množinu, jak je tomu v případě struktur Set a Stream. Stejně jako u seznamu i v množině mohou být obsaženy pouze prvky stejného datového typu.

Množinu vytvoříme analogicky jako seznam, pouze místo hranatých závorek píšeme závorky složené. Pokud napíšeme jakoukoliv hodnotu víckrát, bude zaznamenána pouze jednou. Pro operace s množinami můžeme použít operace **intersect**, **union**, **difference**, **subset?** a další. Použití těchto funkcí je zcela intuitivní, takže nemá smysl je zde dále rozvádět. Dále pak můžeme použít funkce **insert** pro vložení nového prvku do množiny, nebo **remove** pro jeho odstranění.

Jako příklad nehomogenní datové struktury uvedeme Record. Využití tohoto objektu si ukážeme na ukládání informací o zaměstnancích, jejich věku a platech. Vytvoříme proměnnou *daniel* a uložíme do ní údaje o jeho věku a platu. V prvním kroku musíme deklarovat, že proměnná *daniel* bude mít dva prvky se jmény věk a plat typů Integer a Float.

```
(1) -> daniel : Record(vek : Integer , plat: Float)
                                                    Type: Void
```

Do této proměnné můžeme uložit hodnoty tímto způsobem:

```
(2) -> daniel:=[28, 32005.12]
```

(2) $$[vek = 28,\ plat = 32005.12]$$

```
                                  Type: Record(vek: Integer,plat: Float)
```

Pokud chceme změnit údaj o platu, tak provedeme příkaz z dalšího kroku. Obsahem tohoto kroku je i ověření, zda změna proběhla.

```
(3) -> daniel.plat:= 23000;daniel
```

(3) $$[vek = 28,\ plat = 23000.0]$$

```
                                  Type: Record(vek: Integer,plat: Float)
```

Základy symbolických výpočtů

3.1 Práce s proměnnými

Proměnné v systému Axiom se zadávají způsobem, který je patrný s následujícího příkladu:

```
-> xnadruhou:=x**2
```

$$x^2$$

```
                                                    Type: Polynomial Integer
```

Operátor „:=“ reprezentuje přímé přiřazení. Pomocí tohoto operátoru můžeme přiřadit libovolné proměnné hodnotu. Jako jméno proměnné může být použito libovolné označení, kromě výrazů používaných systémem Axiom. Ve jménu proměnné navíc nesmí být použity jiné než alfabetické a numerické znaky. Další druh přiřazení je tzv. „pozdržené“ přiřazení, reprezentované operátorem „==“ a využijeme ho především při definování vlastních funkcí (viz 3.3)

```
-> vyraz:=x**3-y**2
```

$$-y^2 + x^3$$

```
                                                    Type: Polynomial Integer
```

Nyní si ukážeme, jak vypočítat hodnotu proměnné *vyraz* pro konkrétní číselné hodnoty proměnných x a y. Toto se provede pomocí funkce **eval**, jejíž první argument je jméno proměnné, kterou chceme vyčíslit, následovaný seznamem všech přiřazení (v hranatých závorkách, oddělené čárkami), které chceme v symbolickém výrazu provést.

```
-> eval(vyraz,[x=2,y=3])
```

$$-1$$

```
                                                    Type: Polynomial Integer
```

3.2 Řešení rovnic a jejich systémů

Axiom umí řešit rovnice buď exaktně, nebo s použitím určité aproximace. Při přesných výpočtech se iracionální a komplexní kořeny vypisují ve tvaru odmocnin nebo irreducibilních polynomů, zatímco pokud chceme číselný výsledek, tak musíme zadat přesnost aproximace.

Pro řešení jednoduchých rovnic s reálnými kořeny je nejlepší použít funkci **solve**. Jako její první argument se píše rovnice, kterou chceme vyřešit. Druhý argument je nepovinný a reprezentuje proměnnou řešené rovnice. Používá se hlavně pokud rovnice obsahuje parametry. Pokud má rovnice i jiné kořeny než reálné, pak výstup příkazu solve je ve tvaru seznamu reálných řešení plus zbylý irreducibilní polynom:

```
-> solve(x^2=4,x)
```

$$[x = 2, x = -2]$$

```
                Type: List Equation Fraction Polynomial Integer
```

```
-> solve(x**3=8)
```

$$[x = 2, x^2 + 2x + 4 = 0]$$

```
                Type: List Equation Fraction Polynomial Integer
```

Pokud chceme, aby řešení bylo úplné, tedy obsahovalo všechny řešení rovnice, pak máme na výběr dvě možnosti. Můžeme dostat řešení, ve kterém jsou vyjádřeny komplexní kořeny pomocí odmocnin, příkazem **radicalSolve**. Pokud chceme, aby řešení bylo vypsáno přímo formou komplexních čísel, pak se používá příkaz **complexSolve**. Při použití tohoto příkazu se odmocniny aproximativně vypočítají. Tedy u této funkce je povinný druhý argument a udává přesnost, s jakou chceme kořeny aproximovat. Pokud je přesnost zadána formou zlomku tak kořeny budou vyjádřeny také pomocí zlomku. Pokud je přesnost zadána formou desetinného čísla, pak i kořeny budou vyjádřeny pomocí desetinného čísla.

```
-> radicalSolve(x**3=8)
```

$$\left[x = -\sqrt{-3} - 1,\ x = \sqrt{-3} - 1,\ x = 2\right]$$

```
                                    Type: List Equation Expression Integer
```

```
-> complexSolve(x**3=8,0.001)
```

$$\begin{bmatrix} x = -1.0 - 1.73193359375\ i, \\ x = -1.0 + 1.73193359375\ i, \\ x = 2.0 \end{bmatrix}$$

```
                                 Type: List Equation Polynomial Complex Float
```

Systémy rovnic řešíme pomocí stejných funkcí. V prvním argumentu funkce zapisujeme systém rovnic ve formě seznamu (viz 2.4). Tedy celý systém rovnic je ohraničený hranatými závorkami a jednotlivé rovnice jsou odděleny čárkami. Druhý argument je opět zapisován ve formě seznamu neznámých. Pokud chceme vyřešit tento systém rovnic:

$$\begin{aligned} x + y + z &= 8 \\ 3x - 2y + z &= 0 \\ x + 2y + 2z &= 17 \end{aligned}$$

musí příkaz a jeho výsledek vypadat takto:

```
-> solve([x+y+z=8,3*x-2*y+z=0,x+2*y+2*z=17],[x,y,z])
```

$$[[x = -1, y = 2, z = 7]]$$

```
                       Type: List List Equation Fraction Polynomial Integer
```

Systémy rovnic je možné v Axiomu řešit také v maticovém tvaru. Opět se dají použít stejné funkce. Prvním argumentem je matice složená z koeficientů proměnných a druhým argumentem je vektor pravých stran rovnice. Stejná soustava rovnic se dá tedy také řešit:

```
-> solve([[1,1,1],[3,-2,1],[1,2,2]],[8,0,17])
```

$$[particular = [-1, 2, 7], basis = [[0, 0, 0]]]$$

```
                   Type: Record(particular: Union(Vector Frac
tion Integer,"failed"),basis: List Vector Fraction Integer)
```

3.3 Funkce

Systém Axiom dovoluje uživateli definovat si vlastní funkce. Důvody k vytvoření vlastní funkce mohou být dva. První z nich je definování jednoduchých funkcí typu $f(x) = x^3 - 2x$ pokud například funkci chceme používat déle a nechceme se zabývat jejím opakovaným zadáváním. Druhým důvodem je vytváření funkcí, které chceme používat, ale systém Axiom je nemá implementovány. Při tvorbě vlastních funkcí můžeme používat jakékoliv další funkce systému Axiom i vlastní, již dříve vytvořené funkce a další programovací příkazy jako **if, else, repeat, while,...**

Jako příklad tvorby vlastních funkcí ukážeme jen základní možnosti. Definování výše zmíněné funkce a vytvoření funkce pro výpočet faktoriálu (tuto funkci už Axiom obsahuje standardně). Dalších možností při tvorbě funkcí je nespočet a jejich popis s příklady použití lze vyhledat v manuálu k systému Axiom.

Pro definování vlastních funkcí se používá operátor „==“. Jako jméno funkce lze použít libovolný název, který již není použitý pro základní funkce Axiomu, nebo označení makra. Jméno může obsahovat jen alfanumerické znaky. Následující příklad ukazuje definování funkce $f(x) = x^3 - 2x$ a její další použití.

```
(1) -> f==x**3-2*x
                                                    Type: Void
(2) -> f(3)
   Compiling body of rule f to compute value of type
   Polynomial Integer
```

$$(2) \qquad 21$$

```
                                            Type: PositiveInteger
(3) -> solve(f=-1,x)
```

$$(3) \qquad \left[x = 1,\ x^2 + x - 1 = 0\right]$$

```
                Type: List Equation Fraction Polynomial Integer
```

Při tvorbě funkce pro výpočet faktoriálu máme několik možností. Ukážeme si dvě z nich. Obě používají rekurzi. Prvním způsobem lze funkci definovat ve dvou krocích. Nejdříve přiřadíme hodnotu funkce v bodě 0 a v druhém kroku rekurzivní vztah pro výpočet hodnoty funkce v obecném bodě n.

```
(1) -> fact(0)==1
                                                        Type: Void
(2) -> fact(n)==n*(fact(n-1))
                                                        Type: Void
(3) -> fact(15)
```

1307674368000

```
                                                Type: PositiveInteger
```

V druhém způsobu ukážeme, jak stejnou funkci lze definovat v jednom příkazu s použitím konstrukce **if-then-else**.

```
(4) -> fac(n)== if n<3 then n else n*fac(n-1)
                                                        Type: Void
(5) -> fac(15)
```

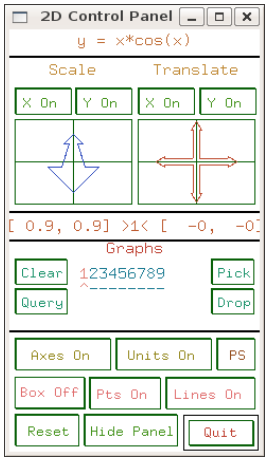
1307674368000

```
                                                Type: PositiveInteger
```

Základy práce s grafikou

V této kapitole se budeme věnovat základům práce s grafikou. Všechny grafické výstupy systému Axiom se vykreslují v nově otevřeném okně, takže je nutné spouštět systém Axiom pod systémem, který toto podporuje. Axiom podporuje 2-dimenzionální i 3-dimenzionální grafiku. Axiom podporuje také polární a sférické systémy souřadnic.

Graf je vykreslován do speciálního okna. Pokud klikneme kurzorem na takto vykreslený graf, otevře se další okno, které obsahuje panel, ve kterém můžeme dodatečně měnit nastavení zobrazení grafu. Vlastností, které můžeme takto měnit, je velké množství a nechybí mezi nimi nastavení měřítka, přiblížení a oddálení libovolné části grafu, zapnutí a vypnutí zobrazení os a další. U 3-dimenzionálních grafů lze navíc libovolně grafem otáčet, měnit barevné zobrazení, nastavit nasvěcování atd. Je zde také možnost uložit graf do formátu ps. Systém uloží soubor do pracovního adresáře systému Axiom.



Obrázek 4.3: Ovládací panel 2-dimenzionálního zobrazení grafu

4.1 2-dimenzionální grafika

Axiom podporuje tyto možnosti dvojrozměrné grafiky:

- funkce jedné proměnné,
- křivky definované parametricky,
- křivky zadané implicitně,
- grafy diskrétních funkcí generované ze seznamu bodů.

Všechny tyto možnosti zobrazuje jak ve standardních souřadnicích, tak i v polárních souřadnicích. Jako příklad 2-dimenzionální grafiky nám poslouží graf funkce jedné proměnné, ostatní tři možnosti jsou podobné a lze je dohledat například v nápovědě k systému Axiom použitím okna Hyperdoc.

4.1.1 Graf funkce jedné proměnné

Obecný tvar příkazu pro vykreslení grafu funkce jedné proměnné $y = f(x)$ je

draw(f(x), x=a..b,*další možnosti nastavení*),

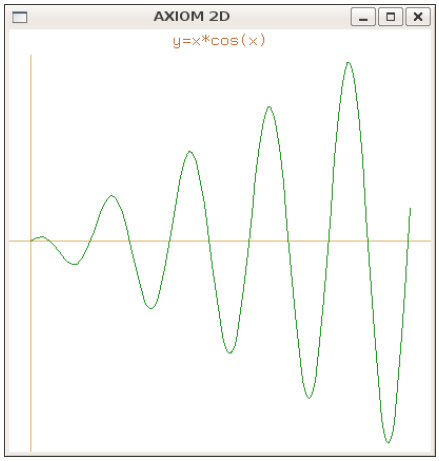
kde a..b je interval, pro které hodnoty x se má graf vykreslit. Část *další možnosti nastavení* je volitelná a může být vynechána (v tomto případě použije Axiom standardní nastavení), nebo může obsahovat i více možností, např. nastavení barvy grafu, určení souřadného systému atd. Možnosti zadání funkce $f(x)$ máme dvě. Přímo do příkazu pro vykreslení grafu, nebo si můžeme nejdřív nadefinovat funkci $f(x)$ pomocí operátoru přiřazení ==. Potom by základní tvar příkazu na vykreslení grafu vypadal takto:

draw(f,a..b)

Než budeme pokračovat dále, ukážeme jeden konkrétní příklad. Požadujeme graf funkce $y = x\cos(x)$ pro hodnoty x v rozmezí 0 až 30. Dále chceme, aby měl graf popisku se zadáním funkce. Zbylé parametry pro nás nejsou důležité, nebo je můžeme následně upravit v ovládacím panelu grafu. V systému Axiom tedy budeme zadávat funkci ve tvaru:

```
-> draw(x*cos(x),x=0..30,title=="y=x*cos(x)")
```

Systém po vyhodnocení předchozího příkazu otevře okno, které můžeme vidět na obrázku 4.4



Obrázek 4.4: Graf funkce jedné proměnné

4.1.2 Některé možnosti nastavení zobrazení dvojrozměrného grafu

Nyní si ukážeme, základní možnosti, kterými můžeme upravit zobrazení grafu. V předchozím příkladu jsme již použili možnost **title==”*text*”**, pomocí které můžeme zadávat název grafu. Všechny tyto možnosti mají stejný syntax jako v použitém příkladu. Další možnosti jsou:

- **curveColor** nastaví barvu, kterou požadujeme pro vykreslení křivky. Základní barvy jsou **red**, **blue**, **green**, **yellow**, ale můžeme si namíchat barvu vlastní použitím operátoru +.
- **pointColor** nastaví barvu bodů.
- **coordinates** pro nastavení souřadnicového systému. Pro polární souřadnicový systém píšeme příkaz ve tvaru **coordinates==polar**.

4.2 3-dimenzionální grafika

Pomocí příkazů pro tvorbu 3-dimenzionální grafiky můžeme tvořit tyto prostorové objekty:

- grafy funkcí dvou proměnných,
- prostorové křivky definované parametricky,
- povrchy definované parametricky.

Narozdíl od dvojrozměrné grafiky nelze vykreslovat grafy zadané implicitně. Je třeba si nejdříve implicitní funkci upravit do explicitního tvaru. Stejně jako u grafů v rovině i 3-dimenzionální grafy můžeme upravovat a měnit souřadnicový systém pomocí různých možností. Jako příklad ukážeme vykreslení grafu funkce dvou proměnných.

4.2.1 Graf funkce dvou proměnných

Pro vykreslení grafu funkce 2 proměnných $z = f(x, y)$ se používá příkaz v obecném tvaru

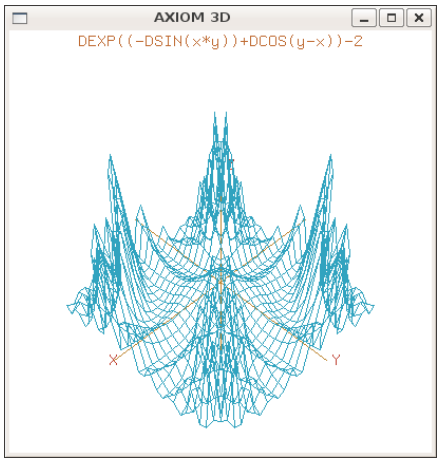
draw(f(x,y), x=a..b,y=c..d,*další možnosti nastavení*),

kde $a..b$ je rozsah hodnot proměnné x a $c..d$ je rozsah hodnot proměnné y, pro které chceme graf zobrazit. Položka *další možnosti nastavení* je nepovinná a nemusí obsahovat žádnou možnost, stejně jako možností několik, analogicky jako u rovinné grafiky.

Pro názornost opět ukážeme použití této funkce na konkrétním příkladu. Chceme vykreslit graf funkce $z = e^{(-\sin(x\ y)+\cos(y-x))} - 2$, bez specifikace dalších parametrů. Příkaz tedy zadáme ve tvaru:

```
-> draw(exp(cos(x-y)-sin(x*y))-2,x=-5..5,y=-5..5)
```

Po vyhodnocení příkazu se otevře okno, které je na obrázku 4.5:



Obrázek 4.5: Graf funkce dvou proměnných

Pokud chceme obrázek uložit do formátu ps, tak na ovládacím panelu (obrázek 4.3) zmáčkneme tlačítko PS. Axiom uloží obrázek do pracovního

adresáře pod názvem *axiom2D.ps*, nebo *axiom3D.ps*, v závislosti na dimenzi ukládaného grafu. Pro změnu názvu je třeba editovat soubor **.Xdefaults** v adresáři, kde je Axiom nainstalován. Uložením dalšího grafu bude předchozí soubor s grafem přepsán, je tedy nutné pro jeho zachování graf buď přejmenovat, nebo přesunout do jiného adresáře.

Některé příklady aplikace Axiomu v matematické analýze

5.1 Limita funkce

Výpočet limity se provádí příkazem **limit**. Jako parametry tohoto příkazu je třeba zadat pouze dva údaje. Těmi údaji jsou: funkce, pro kterou chceme limitu vypočítat a limitní bod proměnné. Dalším volitelným parametrem je směr limity. Pokud třetí parametr nezadáme, tak se Axiom pokusí vypočítat oboustrannou limitu. Nyní si ukážeme jak vypočítat tento výraz:

$$\lim_{x\to 3} \frac{\sqrt{x+1}-2}{x^2-5x+6}$$

Příkaz s výsledkem vypadá takto:

```
(1) -> limit((sqrt(x+1)-2)/(x**2-5*x+6),x=3)
```

(1)
$$\frac{1}{4}$$

```
                                    Type: Union(OrderedCompletion Expression Integer,...)
```

Pokud se hodnota limity liší ve směru zprava a zleva, nebo je definována jenom jedním směrem, nastává situace, ve které můžeme chtít specifikovat směr limity.

Například funkce $\log x$ je definována pouze pro $x > 0$. Proto při výpočtech limit obsahujících tento výraz požadujeme limitu zprava. Výraz

$$\lim_{x\to 0^+} x \log x$$

spočítáme tímto příkazem:

```
(2) -> limit(x*log(x),x=0,"right")
```

(2) 0

```
Type: Union(OrderedCompletion Expression Integer,...)
```

Pokud v tomto příkladu neurčíme směr, tak se Axiom pokusí vypočítat oboustrannou limitu. Protože levostranná limita tohoto výrazu neexistuje, tak snaha o výpočet levostranné limity selže. Axiom nám to dá vědět, nicméně mu to nezabrání provést výpočet pravostranné limity. Z tohoto vyplývá, že specifikace směru limity nebyla nutná, ale pouze zkrátila dobu výpočtu. Toto zkrácení výpočetní doby se projeví při komplikovanějších funkcích.

```
(3) -> limit(x*log(x),x=0)
```

(3) $[leftHandLimit = \texttt{"failed"},\ rightHandLimit = 0]$

```
Type: Union(Record(leftHandLimit: Union(OrderedCompletion
Expression Integer,"failed"),rightHandLimit:
Union(OrderedCompletion Expression Integer,"failed")),...)
```

Nyní si ukážeme jak vypadá výpočet limity, která nabývá odlišných hodnot pro limitu zleva a zprava, bez upřesnění směru.

```
(4) -> limit(sqrt(1-cos(t))/t,t=0)
```

(4) $$\left[leftHandLimit = -\frac{1}{\sqrt{2}},\ rightHandLimit = \frac{1}{\sqrt{2}}\right]$$

```
Type: Union(Record(leftHandLimit: Union(OrderedCompletion
Expression Integer,"failed"),rightHandLimit:
Union(OrderedCompletion Expression Integer,"failed")),...)
```

Nakonec ještě zmíníme, že Axiom ovládá i výpočet limit s parametry a výpočet limit v komplexním oboru pomocí funkce **complexLimit**, která se používá stejně jako funkce **limit**.

5.2 Derivace

Derivování se v Axiomu provádí pomocí funkce **D**. Funkce má v základním tvaru dva parametry. První parametr je derivovaná funkce a druhý parametr je proměnná, podle které chceme derivovat. Funkce **D** umožňuje i výpočet n-té derivace. V tomto případě se jako třetí parametr zadává hodnota n. Pokud například derivujeme funkci f podle proměnné x, tak se funkce zadává ve tvaru **D(f,x)**.

```
(1) -> f:=exp exp x
```

$$(1) \qquad e^{e^x}$$

```
                                                    Type: Expression Integer
```

```
(2) -> D(f,x)
```

$$(2) \qquad e^x\ e^{e^x}$$

```
                                                    Type: Expression Integer
```

Další příkaz vypočítá čtvrtou derivaci stejné funkce f

```
(3) -> D(f,x,4)
```

$$(3) \qquad \left(e^{x4} + 6\ e^{x3} + 7\ e^{x2} + e^x\right)\ e^{e^x}$$

```
                                                    Type: Expression Integer
```

Samozřejmostí v Axiomu je i výpočet parciálních derivací. Používá se stejné funkce **D**, ale jako druhý parametr se udává pořadí proměnných, podle kterých chceme postupně derivace provést. Proměnné se píší do hranatých závorek a musí být oddělené čárkou. První příklad ukazuje výpočet parciální derivace funkce $g(x) = \cos(x^2 + y)$ podle x a druhý příklad parciální derivace stejné funkce v pořadí dvakrát podle y a dvakrát podle x.

```
(4) -> g:=cos(x**2+y)
```

$$(4) \qquad \cos\left(y + x^2\right)$$

```
                                                    Type: Expression Integer
```

```
(5) -> D(g,x)
```

$$(5) \qquad -2\ x\ \sin\left(y+x^2\right)$$

```
                                                    Type: Expression Integer
```

```
(6) -> D(g,[y,y,x,x])
```

$$(6) \qquad 2\ \sin\left(y+x^2\right)+4\ x^2\ \cos\left(y+x^2\right)$$

```
                                                    Type: Expression Integer
```

5.3 Taylorův polynom

Předtím než přejdeme k integraci, věnujeme malý prostor tvorbě Taylorova rozvoje funkcí. V Axiomu k tomuto slouží funkce **taylor**. Funkce má dva parametry. První udává funkci, jejíž Taylorův polynom chceme vypočítat a druhý udává střed Taylorovy řady. Následující dva příklady ukazují výpočet Taylorova polynomu funkce $f(x) = e^{2x}$ se středem v bodě 0 a poté funkce $g(x) = \ln x$ [2] se středem v bodě 1.

```
(1) -> t1:=taylor(exp (2*x),x=0)
```

$$(1)\ 1+2x+2x^2+\frac{4}{3}x^3+\frac{2}{3}x^4+\frac{4}{15}x^5+\frac{4}{45}x^6+\frac{8}{315}x^7+\frac{2}{315}x^8+\frac{4}{2835}x^9+O\left(x^{10}\right)$$

```
                Type: UnivariateTaylorSeries(Expression Integer,x,0)
```

```
(2) -> t2:=taylor(log(x),x=1)
```

$$(2)$$
$$(x-1)-\tfrac{1}{2}\ (x-1)^2+\tfrac{1}{3}\ (x-1)^3-\tfrac{1}{4}\ (x-1)^4+\tfrac{1}{5}\ (x-1)^5-\tfrac{1}{6}\ (x-1)^6+$$
$$\tfrac{1}{7}\ (x-1)^7-\tfrac{1}{8}\ (x-1)^8+\tfrac{1}{9}\ (x-1)^9-\tfrac{1}{10}\ (x-1)^{10}+O\left((x-1)^{11}\right)$$

[2] V Axiomu se přirozený logaritmus zadává jako **log**. K zadání desítkového logaritmu slouží příkaz **log10** a analogicky pro logaritmus o základu 2 atd.

```
                                    Type: UnivariateTaylorSeries(Expression Integer,x,1)
```

Pokud chceme dále použít jenom jeden koeficient této řady, tak ho lze vyvolat pomocí funkce **coefficient**, jak ukazuje následující příklad.

```
(3) -> coefficient(t2,6)
```

$$(3) \qquad -\frac{1}{6}$$

```
                                                    Type: Expression Integer
```

Tímto příkazem jsme získali koeficient šestého členu řady $t2$. Výstup této řady ukazuje jenom prvních deset členů této posloupnosti, ale pomocí funkce **coefficient** můžeme získat libovolný koeficient.

```
(4) -> coefficient(t2,15)
```

$$(4) \qquad \frac{1}{15}$$

```
                                                    Type: Expression Integer
```

Nakonec nutno ještě zmínit, že pokud teď znovu vyvoláme hodnotu proměnné $t2$, tak se vypíše až do patnáctého členu.

5.4 Integrál

K výpočtu integrálu reálné proměnné slouží funkce **integrate**.V základním tvaru se zadává s dvěma parametry, s integrovanou funkcí jako prvním parametrem a proměnnou, podle které má integrace probíhat jako parametrem druhým. Axiom ovládá i integraci v komplexním oboru pomocí funkce **complexIntegrate**, ale touto možností se zde nebudeme zabývat. Bohužel, integrace je v některých případech složitý proces, takže se může stát, že ne všechny integrály bude Axiom schopen vypočítat. Jako první příklad ukážeme výpočet neurčitého integrálu.

```
(1) -> f:=(x+1)/(x-1)
```

$$(1) \qquad \frac{x+1}{x-1}$$

```
                                    Type: Fraction Polynomial Integer
```

```
(2) -> integrate(f,x)
```

$$2\log(x-1)+x \tag{2}$$

Pokud se v integrované funkci nachází parametry, často výsledek integrace záleží na znaménících těchto parametrů. Axiom se uživatele na znaménka neptá a vypočítá výsledky pro všechny možné varianty. Automaticky také předpokládá, že parametry jsou reálná čísla. Pokud chceme připustit možnost komplexního parametru, je nutno integrovat pomocí funkce **complexIntegrate**.

```
(3) -> integrate(1/(x**2+a),x)
```

$$\left[\frac{\log\left(\frac{(x^2-a)\sqrt{-a}+2ax}{x^2+a}\right)}{2\sqrt{-a}},\ \frac{\arctan\left(\frac{x\sqrt{a}}{a}\right)}{\sqrt{a}}\right] \tag{3}$$

```
                          Type: Union(List Expression Integer,...)
```

Pokud Axiom nedokáže vypočítat zadaný integrál v oboru elementárních funkcí, tak se ve výsledku funkce **integrate** objeví znak integrálu.

Nyní postoupíme k výpočtům určitých integrálů. Opět se používá stejné funkce, rozdíl však nastává v druhém parametru, kde se kromě proměnné udává i interval, na kterém chceme integrovat.

```
(4) -> integrate((x**2+1)/3*x,x=1..2)
```

$$\frac{7}{4} \tag{4}$$

```
   Type: Union(f1: OrderedCompletion Expression Integer,...)
```

Další příklad ukazuje, jak Axiom reaguje pokud se v zadaném intervalu vyskytuje hodnota, pro kterou není funkce definovaná. Pokud integrujeme funkci s parametrem a může nastat, že pro nějakou hodnotu parametru není funkce definovaná, Axiom nás na to upozorní a integrál nevypočítá.

```
(5) -> integrate(1/(x-a),x=1..2)
```

(5) $potentialPole$

```
Type: Union(pole: potentialPole,...)
```

Pokud víme předem, že parametr může nabýt takové hodnoty, nebo v reakci na upozornění, můžeme jako třetí parametr funkce **integrate** zadat text **"noPole"**. Tento text Axiomu říká, že hodnoty parametru, ve kterých není funkce definovaná, má vynechat. Axiom v tomto případě integrál vypočítá, ale musíme si být vědomi, že tento výsledek neplatí pro všechny hodnoty parametru. Axiom už na toto dále neupozorňuje.

```
(6) -> integrate(1/(x-a),x=1..2,"noPole")
```

(6) $$\frac{-\log\left(a^2-2\ a+1\right)+\log\left(a^2-4\ a+4\right)}{2}$$

```
Type: Union(f1: OrderedCompletion Expression Integer,...)
```

5.5 Diferenciální rovnice

Protože diferenciálních rovnic je mnoho typů, tak se zde nebudeme zabývat rozebíráním řešení jednotlivých typů těchto rovnic, ale spíše se pokusíme nastínit obecný postup při jejich řešení, který se může mírně lišit typ od typu. Podrobnější rozbor řešení diferenciálních rovnic můžeme najít v materiálech přiložených k distribuci Axiomu.

Řešme nyní diferenciální rovnici třetího stupně $x^3\ y''' + x\ y' - y = 0$. Nechť y je neznámá funkce proměnné x. V prvním kroku deklarujeme y jako operátor[3].

```
(1) -> y:=operator 'y
```

(1) y

```
Type: BasicOperator
```

Dále proměnné *deq* přiřadíme řešenou diferenciální rovnici,

[3]funkce **operator** Axiomu říká, že s nemá s y zacházet jako s proměnnou, ale že jde např. o funkci jiné proměnné.

```
(2) -> deq:=x**3*D(y x, x, 3)+ x*D(y x,x)-y x=0
```

$$x^3\ y'''(x) + x\ y'(x) - y(x) = 0 \tag{2}$$

```
                                    Type: Equation Expression Integer
```

a rovnici vyřešíme příkazem **solve**.

```
(3) -> solve(deq,y,x)
```

$$\left[particular = 0,\ basis = \left[x,\ x\log(x),\ x\log(x)^2\right]\right] \tag{3}$$

```
Type: Union(Record(particular: Expression Integer,basis:
List Expression Integer),...)
```

Tento výsledek nám říká, že obecné řešení této rovnice je ve tvaru $y = C_1x + C_2x\ln x + C_3x\ln^2 x$, kde C_1 ,C_2, C_3 jsou nezávislé parametry. Axiom dále vypočítá jedno partikulární řešení pro hodnoty parametrů rovné 0. V našem případě je partikulárním řešením rovnice $y(x) = 0$.

Někdy je potřeba dostat řešení diferenciální rovnice ve tvaru řady. Proto si na dalším příkladě ukážeme, jak řešit systém dvou nelineárních rovnic prvního řádu tak, aby jsme výsledek dostali ve tvaru řad. K již existující deklaraci y deklarujeme také x jako operátor.

```
(4) -> x:=operator 'x;
                                                  Type: BasicOperator
```

Zadejme dvě rovnice, které tvoří náš systém,

```
(5) -> eq1:=D(x(t),t)=1+x(t)**2
```

$$x'(t) = x(t)^2 + 1 \tag{5}$$

```
                                    Type: Equation Expression Integer
```

```
(6) -> eq2:=D(y(t),t)=x(t)*y(t)
```

$$y'(t) = x(t)\ y(t) \tag{6}$$

```
                                    Type: Equation Expression Integer
```

a soustavu vyřešíme ve formě řad se středem v bodě $t = 0$ s počátečními podmínkami $x(0) = 0$ a $y(0) = 1$. Všimněme si, že pokud zadáme neznámé ve tvaru $[x, y]$, tak dostaneme řešení ve stejném pořadí.

```
(7) -> seriesSolve([eq1,eq2],[x,y],t=0,[x(0)=0,y(0)=1])
```

$$(7) \qquad \left[\begin{array}{l} t + \frac{1}{3}\ t^3 + \frac{2}{15}\ t^5 + \frac{17}{315}\ t^7 + \frac{62}{2835}\ t^9 + O\left(t^{11}\right), \\ \\ 1 + \frac{1}{2}\ t^2 + \frac{5}{24}\ t^4 + \frac{61}{720}\ t^6 + \frac{277}{8064}\ t^8 + \frac{50521}{3628800}\ t^{10} + O\left(t^{11}\right) \end{array} \right]$$

```
   Type: List UnivariateTaylorSeries(Expression Integer,t,0)
```

Závěr

Axiom je plnohodnotný systém počítačové algebry. Během předchozích pár měsíců, kdy jsem měl možnost s tímto systémem pracovat, mě zaujal natolik, že jej budu i nadále používat ve svém dalším studiu aplikované matematiky v kombinaci s ostatním matematickým softwarem.

Systém je výborně ovladatelný textovými příkazy či s pomocí hypertextových odkazů v Hyperdocu. Příkazy jsou snadno zapamatovatelné. Nevýhodou může být absence grafického uživatelského prostředí tak, jak ho má příbuzný systém Maple a při práci v systému Windows nepřítomnost Hyperdocu a vykreslování grafů (toto má být v budoucnu doděláno). Velkou výhodu, oproti Maplu, je však jeho volná šiřitelnost a snažší dostupnost.

Co se týče funkčnosti, tak se domnívám, že oba tyto systémy jsou srovnatelné a vyplatí se Axiom aspoň vyzkoušet a třeba mu i dát přednost před nákupem některého komerčního systému.

Literatura

[1] Jenks, R. D., Sutor R. S.: *Axiom - The Scientific Computation System*, Springer-Verlag, New York 1992. ISBN 0-387-97855-0.

[2] URL: <http://wiki.axiom-developer.org> [cit. 2007-05-15].