PROLOG-1 Tutorium

Wintersemester 2000

Bryan Jurish

Letzte Aktualisierung: 1. Dezember 2000

Inhaltsverzeichnis

1 Accounts (Konten)

- Account = $(Benutzername, Passwort)$ Paar.
- Verwaltet z.B. von:
  - Betriebsysteme,
  - dienstanbietenden Programme.

1.1 Typische Accounts

Account	**Verwaltet von**		**Zweck**
	Rechner	*Programm*	
WinNT	uranos.ling.uni-potsdam.de	OS[1](WinNT)	Einloggen auf Linguistik PCPOOL Rechner
SUN	ling.uni-potsdam.de	OS (Solaris)	Einloggen auf Linguistik SUNPOOL Rechner
E-Mail	kronos.ling.uni-potsdam.de	Maildaemon	Verschicken, empfangen von E-Post über die Linguistik
RZ	rz.uni-potsdam.de	OS (UNIX)	Einloggen auf ZEIK Rechner, E-Post über die ZEIK

1.2 Für PROLOG benötigte Accounts

- Um den PROLOG Compiler / Interpreter zu starten, ist ein SUN Account[2] erforderlich.

[1] OS = "Operating System" (Betriebssystem)

[2] Das SUN OS, "Solaris", ist eine Unterart von UNIX. Deshalb wird innherhalb des Instituts für Linguistik ein SUN Account oft auch "UNIX Account" genannt. Trotzdem sollte man ein SUN Account des Instituts für Linguistik *nicht* mit einem UNIX Account der ZEIK verwirren.

- Um über die PCPOOL Rechner auf die SUNs zuzugreifen wird ferner ein WinNT Account[3] benötigt.
- Anträge für diese Accounts sind sind im Sekretariat oder bei den Pool-Hiwis abzuholen.

2 Typische Vorgänge

2.1 Vorgänge auf PCPOOL Rechner

2.1.1 Einloggen

1. **Voraussetzung:** WinNT Account (§1.2)
2. Rechner wählen, hinsetzen.
3. Falls Rechner noch nicht läuft (grüner LED leuchtet nicht), anschalten (grüner Knopf).
4. Falls nichts auf dem Bildschirm erscheint, Maus bewegen oder Bildschirm anschalten.
5. Beim "Sternenbild", die 3 Tasten Strg, Alt, und Entf gleichzeitig drucken.[4]
6. Im Dialog, WinNT Benutzername im Feld "Login" tippen, und WinNT Passwort im Feld "Passwort" tippen.
7. Weitere Anweisungen auf dem Bildschirm befolgen.

2.1.2 Verbindung zu einem SUN Rechner herstellen

1. **Voraussetzung:** Eingeloggt sein auf PCPOOL Rechner (§2.1.1).
2. Die "Start" Knopf (unten links auf dem Bildschirm) klicken.

[3] "WinNT", oder "Microsnoft Windows NT", läuft typischerweise auf handelsüblichen PCs. Deswegen wird statt "NT Account" auch oft "PC Account" gesagt.

[4] Tastenkombinationen dieser Art werden oft "Strg+Alt+Entf" geschrieben.

3. Die Symbole "Programme, Exceed, Exceed" in dieser Reihenfolge mit der Maus fokussieren; letzeres anklicken. [5]

4. Weiter mit §2.2.1.

2.1.3 Verbindung zu einem SUN Rechner trennen

1. **Voraussetzung:** Eingeloggt sein auf PCPOOL Rechner, der mit einem SUN Rechner verbunden per Exceed verbunden ist (§2.1.2).

2. Von dem SUN Rechner ausloggen (§2.2.8).

3. Beim SUN-Login Dialog, die Tastenkombination "Alt+F4" drucken, um Exceed zu beenden.

2.1.4 Ausloggen

1. **Voraussetzung:** Eingeloggt sein auf PCPOOL Rechner.

2. Alle Benutzerprogramme beenden (§2.1.3 beschreibt diesen Vorgang für Exceed.)

3. Das Symbol "Start → Beenden" anwählen.

4. Im folgenden Dialog, die Option "Anwendungen schliessen ..." wählen, schliesslich "OK" klicken.

2.2 Vorgänge auf SUN Rechner

2.2.1 Einloggen

1. **Voraussetzung:** SUN Account (§1.2)

2. Rechner wählen, hinsetzen.

3. Falls Rechner nicht läuft, in Ruhe lassen!.

[5] Vorgänge dieser Art werden oft "Start → Programme → Exceed → Exceed" geschrieben.

4. Falls nichts auf dem Bildschirm erscheint, Maus bewegen oder Bildschirm anschalten.

5. Beim "Wilkommen" Bildschirm, Sitzungsart wählen[6]; dann Benutzername im Feld "Login" tippen; schliesslich "Enter" (↩) drucken.

6. Im folgenden Dialog, Passwort im Feld "Passwort" tippen; dann "Enter" (↩) drucken.

2.2.2 Starten eines Shells

1. **Voraussetzung:** Eingeloggt sein auf einem SUN Rechner (§2.2.1).

2. **Stark Empfohlen:** Verstehen, was ein Shell ist (§3.1).

3. Die rechte Maustaste auf einem unbestzten Stück Desktop klicken.

4. Im erschienenen Menü "Abeitsbereich Menü", die Einträge "Hosts", und dann "Terminal" anklicken, um den Shellfenster zu öffnen und den Shell zu starten.

5. Andere Möglichkeiten:

   - Arbeitsbereich → Hosts → Konsole
   - Arbeitsbereich → Tools → Terminal

2.2.3 Beenden eines Shells

1. **Voraussetzung:** laufendes Shellfenster auf einem SUN Rechner (vgl. §2.2.2).

2. Auf der Befehlszeile im Shellfenster, "exit" eintippen, gefolgt von "Enter" (↩):

   ```
   bash$ exit ↩
   ```

[6] empfohlen ist das "Common Desktop Environment", "CDE": Optionen → Session → CDE

2.2.4 Starten von SICStus-PROLOG

1. **Voraussetzung:** laufendes Shellfenster auf einem SUN Rechner (vgl. §2.2.2).
2. Auf der Befehlszeile im Shellfester, den Befehl "sicstus" eintippen, gefolgt von "Enter" (↩):

```
bash$ sicstus ↩
```

3. Der SICStus-PROLOG Compiler / Interpreter lädt sich und zeigt einige Informationen an, gefolgt von einem Prompt:

```
bash$ sicstus ↩
SICStus 3.8 (sparc-solaris-2): Feb 16 12:48 2000
Licensed to ling.uni-potsdam.de
| ?-
```

2.2.5 Beenden von SICStus-PROLOG

1. **Voraussetzung:** laufendes SICStus-PROLOG auf einem SUN Rechner (§2.2.4).
2. Beim PROLOG Prompt, den 0-stelligen Prädikat "halt" abfragen:

```
SICStus 3.8 (sparc-solaris-2): Feb 16 12:48 2000
Licensed to ling.uni-potsdam.de
⋮
| ?- halt. ↩
```

2.2.6 Starten eines Editors (xemacs)

1. **Voraussetzung:** laufendes Shellfenster auf einem SUN Rechner (vgl. §2.2.2).
2. Beim Shell-Prompt (Befehlszeile), "xemacs &" eintippen, gefolgt von "Enter" (↩):

```
bash$ xemacs & ↩
```

2.2.7 Beenden von xemacs

1. **Voraussetzung:** laufendes xemacs auf einem SUN Rechner (§2.2.6).
2. Im xemacs Fenster, "File → Exit xemacs" anklicken;
   ODER
   Im xemacs Fenster, die Tastenkombination "Strg+X" drucken, gefolgt von der Tastenkombination "Strg+C".

2.2.8 Ausloggen

1. **Voraussetzung:** Eingeloggt sein auf einem SUN Rechner (§2.2.1).
2. Die rechte Maustaste auf einem unbestzten Stück Desktop klicken.
3. Im erschienenen Menü, "Session Beenden" (???) anklicken.

3 Umgang mit den SUNs

3.1 Shells

3.1.1 Was ist ein Shell?

- Ein Shell ist ein Programm, was einem Benutzer erlaubt, auf die Funktionalität des Betriebssystems (Dateien anlegen, löschen, kopieren; Programme starten, beenden, usw.) zuzugreifen.
- Obwohl es graphische Programme gibt, die Shell-ähnliche Funktionen dienen (z.B. das WinNT Explorer), mit "Shell" ist meistens eine Textbasierte Schnittstelle (Befehlszeile) gemeint.
- Die Programmdatei "command.com" ist z.B. das mitgelieferte Shell für die meisten Microsnoft Systeme.
- UNIX Systeme bieten ein Vielfalt von Shell-Programmen an, z.B. "sh" (Bourne Shell), "bash" (Bourne Again Shell), "csh" (C-Shell), "tcsh" (TC-Shell), usw.

3.1.2 Terminal- bzw. Shell-Fenster

- Heutzutage wird selten ausschliesslich mit einer Textoberfläche gearbeitet – graphische Benutzeroberfächen (GUIs) übernehmen viele Funktionen des klassischen Shells.

- Auch unter GUIs kann es mal nützlich (bzw. nötig) sein, ein klassisches, textbasiertes Shell zu starten. Dieses tut man üblicherweise indem man ein "Fenster" öffnet, in dem das textbasierte Shell läuft.

- Ein Fenster, in dem ein Shell läuft, wird oft "Terminalfenster" oder "Shellfenster" genannt.

- Microsnoft Systeme bieten z.B. oft unter dem Namen "MS-DOS Eingabeaufforderung" ein Shell-Fenster an.

- UNIX Systeme bieten wieder verschiedene Programme, die ein Terminalfenster erzeugen und darstellen können; diese werden z.B. unter den Bezeichnungen "xterm", "cmdtool", "shelltool", "Terminal", "Konsole", oder dem Namen eines Shells selber angeboten.

3.2 Dateien, Verzeichnisse, Pfade: das Dateiensystem

Daten und Programme (sogar physikalische Geräte) sind auf UNIX Systeme über das Dateiensystem erreichbar. Die Hauptbestandteile eines Dateiensystems sind Dateien und Verzeichnisse.

3.2.1 Konventionen

In den folgenden Beispielen wird von den folgenden Konventionen gebrauch gemacht:

- `Text mit fester Breite`
  wird für die wörtliche Darstellung sowohl von der Eingabe von Shell-Befehlen und -Programmaufrufe als auch die Ausgabe des jeweiligen Befehls oder Programms benutzt.

- **Fettschrift**
  wird für die Namen von Shellbefehlen oder Programme in Kurzüberblicken benutzt.

- *Kurisvschrift*
  wird für symbolische Namen der Argumente von Shellbefehlen oder Programmen in Kurzüberblicken benutzt. Wird ein Argumentname zwischen eckigen Klammern gesetzt: *[ArgName]*, ist dieses Argument optional.

Manchmal wird eine (englische) mnemonische Hilfe in ein Kurzüberblick zwischen runden Klammern geschrieben.

3.2.2 Dateien

Eine Datei ("file") kann man als *(Name, Daten)* Paar betrachten. Dokumente und Programme sind Beispiele für Dateien.

3.2.3 Verzeichnisse

Auch als "Ordner" bekannt, Verzeichnisse ("directories") können Dateien oder auch andere Verzeichnisse ("Unterverzeichnisse" bzw. "subdirectories") enthalten (im Gegensatz zu Dateien, die nur "rohe" Daten enthalten dürfen). Das Heimatverzeichnis jedes Benutzers (/home/*Benutzername*) ist ein Beispiel eines Verzeichnisses.

3.2.4 Links, FIFOs, Geräte

Andere Elemente können in einem UNIX Dateiensystem vorkommen: diese sollten wir im Laufe von PROLOG-1 Tutorium aber nicht begegnen.

3.2.5 Navigierung

Um im Dateiensystem zu recht zu finden, können die folgenden Befehle nützlich sein:

- **pwd** ("Present Working Directory")
  Zeigt das aktuelle Verzeichnis an. Unterverzeichnisse werden durch einem Schrägstrich (/) voneinander getrennt. Später wird die Abkürzung "PWD" das aktuelle Verzeichnis bezeichnen.

```
bash$ pwd
/home/moocow/src/prolog
```

- **ls** *[ZielVerzeichnis]* ("List")
  Zeigt den Inhalt vom Verzeichnis *ZielVerzeichnis* an. Wird *Zielverzeichnis* ausgelassen, zeigt **ls** den Inhalt des aktuellen Verzeichnisses an.

```
bash$ ls
aufgabe1.pl   aufgabe12.pl  aufgabe16.pl
aufgabe10.pl  aufgabe13.pl  aufgabe3.pl
aufgabe11.pl  aufgabe14.pl  common.pl
```

- **cd** *[ZielVerzeichnis]* ("Change Directory")
  Macht *ZielVerzeichnis* zum aktuellen Verzeichnis. Ohne Argument wird das Heimatsverzeichnis zum aktuellen Verzeichnis gemacht.

```
bash$ pwd
/home/moocow/src
bash$ cd prolog
bash$ pwd
/home/moocow/src/prolog
bash$ cd
bash$ pwd
/home/moocow
bash$ cd src/prolog
bash$ pwd
/home/moocow/src/prolog
```

3.2.6 Pfade

Dateien und Verzeichnisse können durch *Pfade* im Dateiensystem geortet werden[7]. Pfade können entweder relativ zum aktuellen Verzeichnis oder auch

[7] Diese Art von Pfade sind nicht mit dem DOS Befehl "PATH" zu verwechseln, der genauer "default Pfadliste für ausführbare Programme" genannt werden könnte.

absolut spezifiziert werden. Fast alle Befehle und Programme, die eine Date oder ein Verzeichnis als Argument akzeptieren, nehmen in Wirklichkeit einen Pfad zur Datei oder zum Verzeichnis.

Verzeichnisse, die in einem Pfad vorkommen, werden von ihrem Inhalt durch einem Schrägstrich (/) getrennt.[8]

Ein Pfad ist also eine Liste von Verzeichnisnamen V_i und Dateien D der Form (1) oder (2),

$$V_0/V_1/\ldots/V_n \quad (1)$$
$$V_0/V_1/\ldots/V_n/D \quad (2)$$

wo es gilt:

- V_{i-1} enthält V_i, für alle i mit $1 \leq i \leq n$.
- Für Pfade der Form (2) enthält das Verzeichnis V_n die Datei D.

1. **Absolute Pfade**
   Absolzute Pfade werden vom Wurzel des Dateiensystems ausgehend spezifiziert. Das Wurzelverzeichnis wird "/" gescrieben. Das Wurzelverzeichnis muss nicht durch einem weiteren Schrägstrich von seinen Unterverzeichnissen getrennt werden. Beispiele für absolute Pfade:

   ```
   /user/software/compling
   /bin/ls
   /home/moocow/src/prolog/aufgabe1.pl
   ```

2. **Relative Pfade** Relative Pfade werden relativ zum aktuellen Verzeichnis spezifiziert. In jedem Verzeichnis können zwei besondere Verzeichnisnamen verwendet werden: "." (Punkt) und ".." (Punkt Punkt). Das "." wird durch das zuletzt gelesene Verzeichnis ersetzt (das aktuelle Verzeichnis, falls ein Pfad mit "." anfängt), und das ".." wird durch das Verzeichnis ersetzt, das das zuletzt gelesene Verzeichnis enthält. Das Symbol "~" (tilde) wird durch das absoluter Pfad zum Heimatverzeichnis des Benutzers ersetzt. Beispiele für relative Pfade:

[8] Wichtig! Microsnoft Systeme benutzten anstelle des Schrägstrichs (/) ein Rückwärtsgerichtetes Schrägstrich (\), auch "Backslash" genannt. Das Verwechseln dieser beiden Symbole kann zu unheimlicher Verwirrung und Ärger führen.

`./a.out`	(Programmstart im PWD)
`../a.out`	(Programmstart eine Ebene höher)
`~/src/../tmp`	(= "/home/moocow/tmp")

3.3 Typische Shellvorgänge

3.3.1 Programm starten

Normaleweise werden Programme dadurch gestartet, dass ihre Namen einfach auf die Befehlszeile eingetippt werden, gefolgt von Enter (⏎). Das Befehl:

```
bash$ xemacs ⏎
```

startet z.B. das Programm xemacs.

Liegt ein Programm nicht in einem Verzeichnis im default Pfad für ausführbare Dateien[9], muss einen Pfad (§3.2.6) zum Programm explizit angegeben werden.

3.3.2 Verzeichnisoperationen

- **Erstellen**
  mkdir *NeuesVerzeichnis* ("Make Directory")
  Erstellt im aktuellen Verzeichnis ein Verzeichnis namens *NeuesVerzeichnis*. *NeuesVerzeichnis* kann auch ein beliebiger Pfad sein (§3.2.6).

- **Löschen**
  rmdir *VerzeichnisPfad* ("Remove Directory")
  rm -r *VerzeichnisPfad* ("Remove Recursively")
  Das **rmdir** Befehl entfernt ein leeres Verzeichnis. Das **rm** Befehl mit der "**-r**" Option entfernt ein Verzeichnis zusammen mit seinem gesamten Inhalt. Diese Befehle sind mit Vorsicht zu geniessen.

- **Umbennen / Verschieben**
  mv *AlterPfad NeuerPfad* ("Move")
  Verschiebt die Datei oder das Verzeichnis *AlterPfad* nach *NeuerPfad*.

[9] lässt sich anzeigen mit "echo $PATH"

Falls *NeuerPfad* existiert und ein Verzeichnis ist, wird *AlterPfad* in dieses Verzeichnis hineingeschoben.

Falls *NeuerPfad* nicht existiert, und *AlterPfad* ein Verzeichnis ist, wird der Inhalt von *AlterPfad* nach *NeuerPfad* verschoben; *AlterPfad* wird entfernt.

... usw.

Vorsicht beim verschieben von Dateien und Verzeichnisse, **mv** überschreibt gerne Dateien, ohne nachzufragen!

- **Kopieren**
  cp -r *AlterPfad NeuerPfad* ("Copy Recursively")
  Das Verzeichnis *AlterPfad* wird nach *NeuerPfad* zusammen mit seinem gesamten Inhalt kopiert. Die gleichen Hinweise wie beim **mv** gelten auch hier.

3.3.3 Dateioperationen

- **Erstellen**
  Dateien werden am besten mit Hilfe von Programmen erstellt – z.B. Editoren wie xemacs (§4). Ist es allerdings nötig, eine Datei vom Shell aus zu erstellen, geht das unter anderen mit Hilfe des Programms **touch**:

  touch *DateiName* ()

- **Löschen**
  rm *DateiPfad* ("Remove")
  Die Datei *DateiPfad* wird ohne zu zögern ins ewige Nichts verbannt, ohne Hoffnung auf Wiederkehr.

- **Umbennen / Verschieben**
  mv *AlterPfad NeuerPfad* ("Move")
  Die Datei *AlterPfad* wird nach *NeuerPfad* umbennant/verschoben.

  Falls *NeuerPfad* existiert, und ein Verzeichnis ist, wird die Datei, die *AlterPfad* bezeichnet nach *NeuerPfad* verschoben. Alte Dateien mit dem gleichen Namen werden überschrieben.

  Falls *NeuerPfad* nicht existiert, wird *AlterPfad* in *NeuerPfad* umbenannt.

... usw.

- **Kopieren**
  cp *QuellDateiPfad ZielDateiPfad* ("Copy")
  Die Datei *QuellDateiPfad* wird nach *ZielDatePfad* kopiert. Die gleichen Hinweise wie beim **mv** gelten auch hier.

3.3.4 Online Handbücher Lesen

Die online Handbücher (auch "man-Seiten" genannt) für die meisten Programmen sind mit dem Befehl "man" zu lesen: **man** *ProgrammName* ("Manual)

In xemacs können man-Seiten aufgerufen werden über das "Help" Menü:

Help → Manuals → Unix Manual... → *ProgrammName*

wobei zu beachten ist, dass *ProgrammName* ins Minipuffer (§4.2.4) per Hand getippt werden muss.

Weitere Informationen sind nicht im "man" sondern im "info" Format gespeichert. Diese können in xemacs auch über das "Help" Menü gelesen werden:

Help → Manuals → Info

Die Anfangseite, die dadurch aufgerufen wird, enthält auch als erstes das Link "Info", das Hilfe zum Info-Hilfesystem bietet.

4 Umgang mit dem Editor: xemacs

4.1 Allgemeines

4.1.1 Einrichtung der persönlichen Konfigurationsdatei

Xemacs liest seine Konfiguration aus der Datei "~/.emacs". Eine Vorlage einer solchen Datei, die besondere Konfigurationsbefehle für SICStus-PROLOG enthält, ist in der Datei

```
/user/software/sicstus-emacs/punkt-emacs
```

abgespeichert. Um diese Datei als eigener Konfigurationsdatei zu benutzen, reicht das folgende Befehl:

```
bash$ cp /user/software/sicstus-emacs/punkt-emacs ~/.emacs
```

Hinweis: Xemacs muss nach dem Kopieren der Konfigurationsdatei neu gestartet werden[10]!

4.1.2 Besondere Tastatursequenzen

Xemacs verfügt über Benutzerfreundlichen Knöpfe und Menüs, die einen schnellen Einstieg ermöglichen. Diese Knöpfe und Menüs funktionieren allerdings nicht immer so, wie man erwartet. Deswegen ist der Umgang mit den besonderen Tastatursequenzen (auch "Steuerungs-Sequenzen" genannt, weil der Benutzer das xemacs Programm selber damit steuert) von xemacs manchmal nötig.

Steuerungs-Sequenzen werden durch Abkürzungen bezeichnet, die alle Tasten auflisten, die ein Benutzer drucken muss, um den assozierten Vorgang auszulösen. Diese Abkürzungen haben immer die folgende Form:

Modifizierer - Taste [[Modifizierer-] Taste]

Das zweite Sequenz (*[[Modifizierer-] Taste]*) ist Optional, und falls sie vorkommt, ist ihr *Modifizierer* Teil auch Optional.

Mögliche Werte für *Modifizierer* in Abkürzungen sind die Zeichen 'C' und 'M', wobei das 'C' "Control" ("Steuerung") abkürzen soll, und das 'M' für "Meta" darsteht. Die Sequenz "C-g", zum Beispiel, wird "Control-g" gelesen, und die Sequenz "M-w" wird "Meta-w" oder "Escape-w" gelesen.

Der 'C' Modifizierer ist die Strg Taste. Bei Sequenzen der Form "C-g" wird die Strg Taste gedrückt gehalten, während dessen die G Taste gedruckt wird. Die Microsnoft-typische Abzeichnung einer solchen Abkürzung ist Strg+G.

[10] nicht wirklich, aber der Vorgang, um die Datei einfach neu zu lesen, ist etwas komplizierter als ich im Moment beschreiben möchte.

Wichtig: Xemacs Tastatursequenzen sind gross-/klein-sensitiv, also"C-g" $\neq$ "C-G"!

Der 'M' Modifizierer ist, laut Xemacs Tutorial[11] die "ALT" Taste einer PC, oder "auf einer Sun Tastatur die Raute-Taste links vom SPACE Balken". Demzufolge ist eine Abkürzung "M-w" dadurch auszuführen, dass die ALT Taste gedrückt gehalten, während dessen die W Taste gedrückt wird. Der Autor persönlich neigt aus verschiedenen Gründen dazu, die "ESC" Taste als Meta-Taste zu benutzen – diese Neigung führt aber dazu, dass er für die Abkürzung "M-w" zuerst Esc drücken und loslassen muss, und erst dann w tippt.

4.1.3 Wichtige Tastatursequenzen

In der folgenden Tabelle sind manche nützliche Tastatursequenzen von xemacs aufgelistet:

Sequenz	**Beschreibung**
C-g	Abbrechen des aktuellen Vorgangs
C-x C-s	Speichern des aktuellen Puffers
C-x C-w	Speichern des aktuellen Puffers unter einem neuen Namen
C-x C-f	Öffnen einer Datei in einem neuen Puffer
M-g	Kursor zur bestimmten Zeile bringen
C-SPACE	Text durch Kursorbewegung markieren
C-w	Ausschneiden des markierten Textes
M-w	Kopieren des markierten Textes
C-y	Einfügen des zuletzt kopiert oder ausgeschnittenes Textes an der Kursorposition
C-x C-c	Xemacs beenden

4.2 Das Xemacs Fenster

Abbildung 1 zeigt ein typisches xemacs Fenster. Wichtige Bestandteile dieses Fensters sind ettikiert. Diese Bestandteile werden in den folgenden Abschnitte kurz beschrieben.

[11] Help → Basics → Tutorials → *IrgendEineSprache*

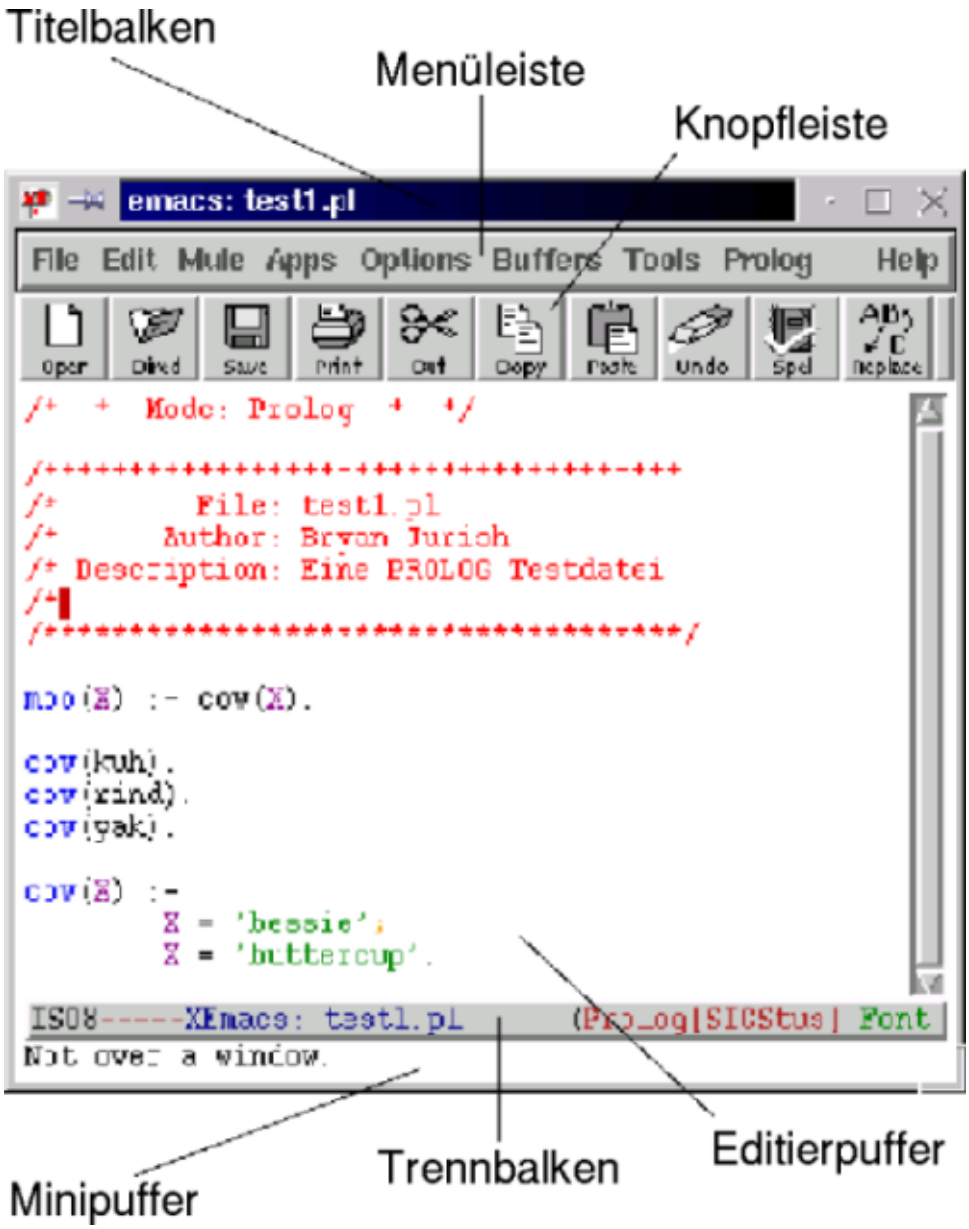

Abbildung 1: Ein Xemacs Fenster

4.2.1 Menü- und Knopfleisten

Diese Leisten erlauben den Benutzer, häufig durchgeführte Vorgänge mit der Maus auszulösen, durch klicken auf ein Symbol (Knopfleiste) oder auf einen Menüeintrag (Menüleiste) für den gewünschten Vorgang. Menüs sind hierarchisch orgranisiet – d.h. eine Menü (z.B. "Options") darf auch ander Menüs enthalten.

4.2.2 Puffer

Das wichtigste Bestandteil eines Xemacs Fensters sind die sogenannten *Puffer* ("Buffers"). Jedes Puffer trägt zu Identifizierung einen eindeutigen Namen. Ein Puffer kann unter anderen für eins der folgenden Zwecke benutzt werden:

- **Dateien editieren: Dateipuffer**
  Ein sog. *Dateipuffer* ("file buffer")[12] kann eingesetzt werden, um den Inhalt einer Datei innerhalb von xemacs anzuzeigen, und zu ändern. Dateipuffer sind die am häufigsten vorkommenden Art von Puffer. Der Name eines Dateipuffers gleicht normaleweise dem Namen der darin zu editierender Datei.

- **Prozesse steuern: Prozesspuffer**
  Beliebige Programme können innerhalb von xemacs gestartet werden[13] Ein solches Programm besitzt ein eigenes Puffer, von dem es seine (Text-)Eingabe liest, und zu dem es seine (Text)-Ausgabe schreibt. Diese *Prozesspuffer* (auch "Programmpuffer" genannt) tragen üblicherweise den Namen "**ProgrammName**". SICStus-PROLOG, z.B., lässt sich über das "Prolog" Menü in einem (neuen) Puffer namens "*prolog*" starten.

- **Dokumentation lesen: Info-Puffer**
  Dokumentation im "info" Format[14] kann innerhalb von xemacs gelesen werden. Die Dokumentation, z.B., die über der "Help" Menü zur Verfügung steht, ist hauptsächlich in diesem Format. Hilfedateien werden in einem eigenen Puffer namens "*info*" angezeigt.

[12] auch "Editierpuffer" genannt
[13] Besonders nützlich sind Programme mit Textbasierten Ein- und Ausgaben.
[14] Das "info" Format ist aus Benutzersicht HTML sehr ähnlich.

- **Notizen schreiben: das *scratch*-Puffer**
  Bei seinem Start öffnet xemacs per Default ein Puffer namens *scratch*, der in den meisten Hinsichten wie ein Dateipuffer funktioniert. Dieser Puffer ist dafür gedacht, Notizen zu enthalten, die nicht in eine Datei gespeichert werden sollen. Umbenennen des *scratch*-Puffers (z.B. durch einen "Save As" Vorgang) können zu unerwarteten Resultaten führen.

- **Das aktuelle Puffer**
  Das "aktuelle" Puffer zu einem bestimmten Zeitpunkt ist dasjenige Puffer, in dem sich das Kursor befindet. Viele Operationen von xemacs – sogar die Besatzung der Menüleiste, hängen von der Art des aktuellen Puffers ab. Der Name des aktuellen Puffers steht normaleweise im Titelbalken.

4.2.3 Trennungsbalken

Trennungsbalken zeigen Informationen bezüglich desjenigen Puffers an, das direkt über ihnen im xemacs Fenster steht. Die angezeigte Informationen enthalten z.B.

- den Namen des Puffers,
- das Modus des Puffers (z.B. "Prolog[SICStus]" – eine Unterart eines Dateipuffers), und
- Zeile- und Spaltenummer der aktuellen Kursorposition.

Hinweis: Wie fast alles in xemacs ist die im Trennungsbalken angezeigte Information individuell einstellbar. Details sind im xemacs Hilfesystem zu finden.

4.2.4 Das Minipuffer

Das Minipuffer wird von xemacs benutzt, um Benutzereingaben für textbasierte Vorgänge zu lesen, und um Nachrichten über durchzuführende Operationen an den Benutzer auszugeben. Das Suche- und Ersetzungsverfahren,

z.B. liest sowohl die zu suchende als auch die einzuführende Zeichenkette aus dem Minipuffer.

Bei Vorgängen, die das Lesen von Benutzereingaben aus dem Minipuffer erforden, wird das Minpuffer automatisch zum aktiven Puffer gemacht – der Kursor wird ins Minpuffer gesetzt. Sollte ein Benutzer im Laufe eines Minipuffer-Lesevorgangs einen anderen Puffer aktivieren (dadruch, dass er/sie auf das andere Puffer klickt), bricht das den Minipuffer-Lesevorgang **nicht** ab. Eine Symptom dieses Problems ist das erscheinen der folgenden Nachricht im Minipuffer:

```
Minibuffer already active ... enable a new one with 'n'
```

Erscheint diese Nachricht im Minipuffer, ist wie unten angegeben vorzugehen:

1. Mit der Maus in das Minipuffer klicken, um es zu aktivieren.
2. Die Tastensequenz C-g (Strg+G, vgl. §4.1.2) drücken, um den störenden Minipuffer-Lesevorgang abzubrechen.

Hinweis: Wenn etwas nicht funktioniert, Inhalt des Minipuffers lesen!

4.3 Menüs

4.3.1 File

- **Open**
  Öffnet eine Datei ein einem neuen Puffer. Falls die angegebene Datei noch nicht existiert, wird ein Puffer trotzdem für sie erzeugt. Die Datei wird erzeugt sobald die (nicht-leere) Puffer gespeichert wird.

- **Save**
  Speichert das aktuelle Puffer in einer Datei. Ist noch keine Datei mit dem Puffer assoziert (z.B. der Puffer ist das Notizpuffer "*scratch*" oder ein Prozesspuffer wie z.B. "*prolog*"), muss einen Dateinamen angegeben werden.

- **Save As**
  Speichert das aktuelle Puffer unter einem neuen Namen, der angegeben werden muss.

- **Split Window**
  Trennt das Xemacs Fenster so, dass zwei Puffer anstelle des aktullen Puffers sichtbar werden. Kann für das austesten von PROLOG Programmen sehr nützlich sein.

- **Un-Split**
  Sind zwei Puffer sichtbar, macht ein "Un-Split" nur einen davon sichtbar.

- **Revert Buffer**
  Lädt die Datei, die mit dem aktuellen Puffer assoziert ist von der Festplatte neu ein. Änderungen des Puffers seit dem letzten Speichervorgang gehen verloren.

- **Delete Buffer**
  Schliesst das aktuelle Puffer. Das "Delete" bezieht sich jedoch nur an Xemacs – wenn es eine Datei gibt, die mit dem Puffer assoziert ist, wird diese **nicht** durch das anwählen von "Delete Buffer" gelöscht.

- **Exit XEmacs**
  Schliesst das Xemacs Fenster und alle Puffer, die derzeit geöffnet sind.

4.3.2 Edit

- **Undo**
  Macht den letzten Editiervorgang rückgängig.

- **Cut**
  Schneidet das derzeit markierte Text aus dem aktuellen Puffer heraus.

- **Copy**
  Kopiert das derzeit markierte Text von dem aktuellen Puffer.

- **Paste**
  Fügt das zuletzt kopierte oder ausgeschnitte Text in das aktuelle Puffer an der Kursorposition ein.

- **Search**
  Sucht im aktuellen Puffer von der Kursorposition ausgehend bis zum Ende des Puffers nach einer Zeichenkette. (die gesuchte Kette ist in das Minipuffer (§4.2.4) zu tippen)

- **Replace**
  Sucht und ersetzt Zeichenketten im aktuellen Puffer von der Kursorposition ausgehend bis zum Ende des Puffers. (die Ketten sind in das (§4.2.4) zu tippen!)

4.3.3 Buffers

- **List All Buffers**
  Schreibt in ein (neues) Puffer namens "*Buffer List*" eine Liste alle derzeit geöffneten Puffer, zusammen mit deren Grössen, Modi, und assozierten Dateien.

- **(Puffer-Name)**
  Die Namen alle derzeit geöffneten Puffers erscheinen auch im "Buffers" Menü. Das anwählen eines Puffers aus dieser Liste bringt das angewählte Puffer "nach vorne" – das heisst, das angewählte Puffer wird an der Stelle des aktuellen Puffers gebracht.

4.3.4 Prolog

Wird eine PROLOG Datei in einem Puffer editiert, erscheint in der Menüleiste eine Menü namens "Prolog".[15] Diese Menü enthält u.a. folgende Einträge:

- **Consult**
  - **File**
    Lässt die mit dem aktuellen Puffer assozierte Datei in das (neue) SICStus-PROLOG Prozesspuffer "*prolog*" einlesen. Hat das gleiche Effekt wie die Eingabe von

    `?- consult('`*Datei*`').` ↩

[15] Das erscheinen dieser Menü ist Modus-abhängig: falls das Menü *nicht* erscheint, siehe §4.1.1.

oder

```
?- ['Datei'].
```

in den SICStus-PROLOG Interpreter.

Wichtiger Hinweis: Nur die zuletzt gespeicherte Version der Datei wird durch "Prolog → Consult → File" konsultiert! Generell gilt: abspeichern ("File → Save") bevor konsultieren!

- **Buffer**
  Der Inhalt des aktuellen Puffers wird vom (neuen) SICStus-PROLOG Prozesspuffer "*prolog*" konsultiert.
- **Region**
  Die derzeit markierten Region des aktuellen Puffers wird vom (neuen) SICStus-PROLOG Prozesspuffer "*prolog*" konsultiert.
- **Predicate**
  Das PROLOG Prädikat, in dessen Definition das Kursor des aktuellen Puffers sich befindet, wird vom (neuen) SICStus-PROLOG Prozesspuffer "*prolog*" konsultiert.

Hinweis: "das (neue) *prolog* Puffer" heisst, dass ein PROLOG Prozesspuffer namens *prolog* erzeugt wird, falls noch keins existiert. Falls ein *prolog* Puffer schon existiert, wird dieses auch benutzt.

- **Run Prolog**
  Startet ein neues SICStus-PROLOG Prozesspuffer in dem Puffer namens "*prolog*".
- **Help on Predicate**
  Das SICStus-PROLOG Handbuch für das Benutzerspezifierte Prädikat wird in dem Info-Puffer angezeigt.

4.3.5 Help

Über die "Help" Menü sind verschiedene Dokumentationen erreichbar, unter anderen das Handbuch von SICStus-PROLOG.

- **Help → Basics → Tutorials**
  Weitere Informationen bezüglich xemacs sind in verschiedenen Sprachen hier zu finden.

- **Help → Lookup in Info → Topic → "sicstus"**
  Zeigt das SICStus-PROLOG Handbuch.

- **Help → Recent Messages**
  Zeigt die letzten Minipuffer-Nachrichten ein einem Puffer "*Help: lossage*" an.

5 Umgang mit SICStus-PROLOG

5.1 Kompilierung vs. Interpretierung

SICStus-PROLOG, wie alle mir bekannten PROLOG Implementierungen, besteht aus 2 Komponenten: aus einem *Kompilierer*[16] und einem *Interpreter*.

5.1.1 Kompilierung: Programmquellen Konsultieren

Um etwas nützliches mit PROLOG zu machen, muss in der Regel zuerst eine Sammlung von Fakten und Regeln erstellt werden. Diese wird normalerweise in einem Editor erstellt – beispielsweise in xemacs (vgl. §4) – und in eine Datei (vgl. §3.2) abgespeichert. Dateien, die PROLOG Programmquelltext enthalten, werden nach Konvention mit Namen gespeichert, die mit ".pl" enden.

Um PROLOG auf die selbstdefinierten Fakten und Regeln in der Quelldatei aufmerksam zu machen, muss diese Datei vom laufenden PROLOG Prozess *konsultiert* werden – durch Konsultierung einer Datei werden die Ihalte der Datei zur PROLOG Datenbasis hinzugefügt.

Statt "konsultieren" einer Quelldatei, sagt man oft auch "*Kompilierung*" der Datei – "kompilieren" heisst, für Menschen lesbare Quelltexte in eine für den Rechner einigermassen effizient verarbeitbare Form bringen. Kompiliereung einer Datei verändert **nicht** den ursprünglichen Inhalt der Datei – die Maschinenrepräsentation, die bei der Kompilierung entsteht, behält PROLOG für sich.

Von xemacs aus können Dateien, markierte Regionen, und Pufferinhalte über

[16] engl. "Compiler"

das "Prolog" Menü von dem PROLOG Prozess im "*prolog*" Puffer konsultiert werden (vgl. §4.2.2, §4.3.4).

Möchte man eine Datei konsultieren, wenn die xemacs Menüs nicht zur Verfügung stehen – z.B. von einem Shell (§3.1) aus, über einer langsamen Netzverbindung, oder sogar automatisiert von einer anderen PROLOG Quelldatei aus, kann entweder das eingebaute PROLOG Prädikat `consult/1` oder das eckige Klammer Operator[17] [...] eingesetzt werden:

- Vom PROLOG-Prompt aus, die Datei 'test.pl' im aktuellen Verzeichnis mittels des Prädikates `consult/1` konsultieren:

  ```
  ?- consult(test). ↩
  ```

- Vom PROLOG-Prompt aus, die Datei 'test.pl.debug' im aktuellen Verzeichnis mittels des Prädikates `consult/1` konsultieren:

  ```
  ?- consult('test.pl.debug'). ↩
  ```

- Vom PROLOG-Prompt aus, die Datei 'test.pl' im aktuellen Verzeichnis mittels des eckigen Klammer Operators konsultieren:

  ```
  ?- [test]. ↩
  ```

- Vom PROLOG-Prompt aus, die Datei 'test.pl.debug' im aktuellen Verzeichnis mittels des eckigen Klammer Operators konsultieren:

  ```
  ?- ['test.pl.debug']. ↩
  ```

- Vom PROLOG-Prompt aus, die Datei 'test.pl' im Unterveichnis 'debug/' des aktuellen Verzeichnises mittels des eckigen Klammer Operators konsultieren:

  ```
  ?- ['debug/test.pl']. ↩
  ```

- Von einer PROLOG Quelltextdatei aus, die Datei 'test.pl' mittels des eckigen Klammer Operators konsultieren:

  ```
  :- [test]. ↩
  ```

[17] engl. "square brackets"

5.1.2 Interpretierung: Anfragen

Läuft ein PROLOG Prozess, z.B. in einem Shellfenster (§3.1.2) oder in einem xemacs Puffer (§4.2.2), wird einen Prompt gezeigt (in SICStus-PROLOG, "?- "), nach dem ein Benutzer dem PROLOG Prozess mit der Tastatur Anfragen stellen kann, z.B. eine `consult/1` Anfrage, die eher eine Anweisung zum kompilieren ist (§5.1.1).

Am PROLOG Prompt gestellte Anfragen werden von PROLOG **nicht** in der gleichen Weise verarbeitet wie konsultierte Programmquelltexte, obwohl Anfragen und Programmquelltexte aus den gleichen Bestandteile (Fakten, Regeln, Terme, Logischen Operatoren, usw.) aufgebaut sind.

Anfragen werden statt von dem Compiler, von dem PROLOG *Interpreter* verarbeitet. Wird eine Anfrage gestellt, versucht der PROLOG Interpreter für diese Anfrage einen Beweis zu finden, mittels seiner Datenbasis aus Fakten, Regeln, usw. Wird ein Beweis gefunden, antwortet PROLOG mit "Yes"; ist die Anfrage nicht beweisbar, antwortet PROLOG dagegen "No".

Kommt eine Variable in einer Anfrage vor, versucht PROLOG die Anfrage für *mindestens eine Instantiierung* dieser Variable zu beweisen. Ist dies gelungen, zeigt PROLOG die Instantiierungen aller in der Anfrage enthaltenen Variablen an, zusammen mit einem Prompt ("?"), um zu erfragen, ob mittels Backtracking weitere mögliche Instantiierungen der Variablen gesucht werden sollen, für die die gestellte Anfragen ebenfalls beweisbar ist. Tippt ein Benutzer bei diesem Prompt das Semikolon (";"), such PROLOG nach weitere Instantiierungen. Tippt der Benutzer stattdessen Enter (⏎), hört PROLOG mit dem Beweisverfahren an der Stelle auf, und gibt die Antwort "Yes" aus. Hat der Benutzer Semikolon getippt, und ist die Anfrage für keine weitere Variableninstantiierungen beweisbar, antwortet PROLOG mit "No".

5.2 Trace und Debug

Die Details des PROLOG Beweisverfahrens kann der Benutzer sich anzeigen lassen mit Hilfe des Prädikates `trace/0`. Dieses Prädikat ist immer beweisbar, und kann als eine Anweisung, die Beweisinnereien anzuzeigen, verstanden werden. Falls eine Anfrage unbeweisbar bleibt, für die einen Beweis eigentlich existieren müsste, oder falls eine Anfrage beweisbar ist, die das nicht sein sollte, kann die Ausgabe vom `trace` Modus behilflich sein.

Die Ausgabe in `trace` Modus[18] Zeigt eine Zeile für jeden Schritt des Beweisverfahrens an. Argumente, die mit einem Unterstrich ("_") anfangen, und sonst nur aus Zahlen bestehen, stellen uninstantiierte Variablen da. In xemacs wird in `trace` Modus die für das Beweisverfahrenschritt aktuelle Zeile in der Quelldateipuffer farblich hevorgehoben. Der nächste Schritt wird angezeigt, wenn beim PROLOG `trace` Prompt ("?"), Enter getippt wird. Weitere Details zur `trace` sind in der SICStus-PROLOG Dokumentation zu finden (§4.3.5).

`trace` Modus lässt sich vom PROLOG Prompt wieder ausschalten mit einer Anfrage an das Prädikat `notrace/0`.

Beispiel Quelldatei 'cows.pl':

```
moo(X) :- cow(X).
cow(kuh).
cow(rind).
cow(yak).
```

Beispiel Interpreter-Ablauf:

```
| ?- [cows].   % Konsultieren von 'cows.pl'
{consulting /home/moo/prolog/cows.pl...}
{consulted /home/moo/prolog/cows.pl in module user}

yes
| ?- trace.
{The debugger will first creep -- showing everything}

yes
| ?- moo(X).
        1      1 Call: moo(_196) ?
        2      2 Call: cow(_196) ?
?       2      2 Exit: cow(kuh) ?
?       1      1 Exit: moo(kuh) ?

X = kuh ? ;
        1      1 Redo: moo(kuh) ?
        2      2 Redo: cow(kuh) ?
?       2      2 Exit: cow(rind) ?
?       1      1 Exit: moo(rind) ?
```

[18] auch als `debug` Modus bekannt

```
X = rind ? ;
        1      1 Redo: moo(rind) ?
        2      2 Redo: cow(rind) ?
        2      2 Exit: cow(yak) ?
        1      1 Exit: moo(yak) ?

X = yak ? ;

no
```

6 Andere Informationsquellen

1. Die Kursmaterialien von Thomas Hanneforth sind über die HTML Seiten des Instituts für Linguistik erreichbar. Diese enthalten umfangreiche Informationen zu PROLOG, inklusiv eine Literaturliste:

   http://www.ling.uni-potsdam.de/kvv/ws00/Prolog_fuer_Linguisten.html

2. Michael Goetze hat ein weiteres Skript (im HTML Format) geshrieben, das ein Überblick über UNIX Befehle, xemacs, und SICStus-PROLOG bietet:

   http://www.ling.uni-potsdam.de/~goetze/prolog

   Dieses Skript ist auch über die PROLOG-1 Seite (1) verfügbar.

3. Neue Versionen dieses Dokuments sind in verschiedenen Formaten unter dem folgenden URL vergübar:

   http://www.ling.uni-potsdam.de/~moocow/classwork/pl1

   Diese Seite ist auch über die PROLOG-1 Seite (1) verfügbar.

4. Die TU München bietet eine sehr Umfangreiche Online Einführung in UNIX:

   http://www.informatik.tu-muenchen.de/rechenbetrieb/documents/anleitung/anl.toc.html#3

5. Die beste Quelle für Informationen zu den UNIX Befehlen sind die online Handbücher, mit dem `man` Programm im Shell zu lesen:

   `bash$ man` *ProgrammName*

6. Die beste Quelle für Informationen zu dem Shell ist die Dokumentation des Shells selber, mit dem Shell Befehl `help` aufzurufen:

   ```
   bash$ help
   ```

   Andere wichtige Informationen zu dem Shell sind in den man-Seiten (Nr. 5, diese Abschnitt) zu finden.[19]

7. Das Xemacs Hilfesystem ist über die "Help" Menü (§4.3.5) im xemacs verfügbar. Dieses bietet sowohl Zugriff auf die UNIX man-Seiten als auf die Systemdokumentation im "info" Format, inklusiv die Dokumentation zu SICStus-PROLOG und zu xemacs selber.

[19] *ProgrammName* = "bash"