Введение в ИИ на примере языка Prolog

Файловые операции

https://github.com/Inscriptor/IntroductionToAI/tree/master/pdf

Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»

20 октября 2019 г.

Разделение программ и модульность

Разделение программ и модульность

Переиспользование кода

Чтобы указать интерпретатору о необходимости предварительно прочитать дополнительный код, в начало программы следует добавить список требуемых файлов:

```
:- [source1, source2, ...]
```

Данная инструкция заставит интерпретатор сначала обработать (**consult**) все файлы из заданного списка, а затем прочитать текущий файл.

Разделение программ и модульность

Переиспользование кода

Чтобы указать интерпретатору о необходимости предварительно прочитать дополнительный код, в начало программы следует добавить список требуемых файлов:

```
ensure_loaded([source1, source2, ...])
```

Предикат `ensure_loaded/1` для каждого указанного файла в списке проверяет, был ли он прочитан ранее, и если да, то изменилось ли что-нибудь в нем. Если файл не был прочитан, или с момента последнего чтения был изменен — происходит его компиляция. Рекомендуется всегда использовать данный предикат вместо более простой предыдущей записи.

Разделение программ и модульность

Модули и экспорт

- Допустим, у нас имеется 2 файла `source1.pl` и `source2.pl`, в которых определены нужные нам предикаты `func1` и `func2` соответственно. Допустим, также, что в каждом из файлов определен свой предикат `auxFunc`, от которого зависят предикаты `func1` и `func2`.
- Если мы попытаемся загрузить файлы предыдущим способом, то, скорее всего, мы получим сообщение об ошибке. Либо, что еще хуже, определение предиката `auxFunc` из файла `source2.pl` затрет предыдущее определение, что может привести к ошибкам в работе предиката `func1`.
- Нашу текущую программу не интересуют зависимости предикатов `func1` и `func2`.

Разделение программ и модульность

Модули и экспорт

Вместо полной обработки файлов `source1.pl` и `source2.pl` мы можем объявить их **модулями** и экспортировать те предикаты, которые должны быть видны извне.

```
:- module(Name, ListOfPredicatesToExport).
:- module(mod1, [func1/1]).
```

Модульность позволяет скрыть определения предикатов, которые нужны только внутри модуля. Предикаты, которые могут быть вызваны снаружи модуля, называются *публичными (public)*, остальные — *приватными (private)*.

Разделение программ и модульность

Модули и экспорт

Чтобы **импортировать** модуль, необходимо в начало программы добавить вызов предиката `use_module`.

```
:- use_module(moduleName).
:- use_module(source1).
```

Предикат `use_module/1` импортирует все public-предикаты из модуля.

Разделение программ и модульность

Модули и экспорт

Чтобы **импортировать** модуль, необходимо в начало программы добавить вызов предиката `use_module`.

```
:- use_module(moduleName, ListOfPredicatesToImport).
:- use_module(source1, [func1/1]).
```

В то время, как `use_module/2` позволяет указать список предикатов, которые необходимо импортировать.

Разделение программ и модульность

Библиотеки

- Если при импорте указать, что загружаемый модуль является библиотекой, интерпретатор будет искать его не в текущей директории, в месте хранения библиотек.
- `:- use_module(library(lib)).`
- SWI-Prolog предоставляет множество встроенных предикатов и дополнительных библиотек, в том числе для работы с графическими компонентами.
- Другие реализации, например Sicstus, не имеют встроенных предикатов, а весь вспомогательный инструментарий реализован в виде библиотек.
- В различных реализациях Пролога один и тот же функционал может быть реализован по-разному, поэтому, если требуется, чтобы программа работала одинаково вне зависимости от интерпретатора, рекомендуется не полагаться на библиотеки и по возможности реализовывать критически важный функционал самостоятельно.

Чтение и запись файлов

Чтение и запись файлов

Открыть файл, закрыть файл

Прежде, чем можно будет приступить к работе с файлом, его необходимо открыть при помощи предиката `open/3`.

```
open(+FileName,+Mode,-Stream)
```

Предикат принимает 2 параметра и отдает 1 в качестве результата:

- `FileName` — имя файла. Ничего необычного.
- `Mode`. Режим работы с файлом: `read`, `write`, `append`.
- `Stream`. Уникальный идентификатор потока, присвоенный Прологом.

В случае открытия файла в режимах `write` или `append` если файла с заданным именем не существует — он будет создан.

Чтение и запись файлов

Открыть файл, закрыть файл

В целом, сеанс связи с файлом выглядит примерно так:

```
open(myfile,write,Stream),
...
do something,
...
close(Stream),
...
```

Чтение и запись файлов

Чтение и запись

Для записи в файл используются те же предикаты, что и для вывода в консоль: `write, writeln, tab, nl`. Только на этот раз первым параметром приходится указывать идентификатор потока файла.

```
writeFile :- open('out.txt',write,Out),
             write(Out,'Hello,World!'), close(Out).
```

Очевидно, что первым параметром предиката `open/3` может быть и путь к файлу. Также, может понадобиться предикат `working_directory/2`, если требуется сменить текущую рабочую директорию.

Чтение и запись файлов

Чтение и запись

Чтение осуществляется предикатом `read/2`.

```
readFile :- open('in.txt',read,In),
            read(In, Input),
            writeln(Input),
            close(Out).
```