Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

EARIN

Laboratory 7
Group 21
Variant 1
Logic and Inference

Tymon Żarski 310992
Bartosz Peczyński 310703

WARSAW May 23, 2024

Contents

1. Introduction

1.1. Assigned task

Solve one of the following problems using Prolog.

Variant 1

Perform simple addition and subtraction of dates by a number of N days. Assumptions:

- The year is 2024
- $N \leq 366$

Example:

```
?- add_date("2205", 14) % add 14 days from 22nd of May
"0506" % 5th of June

?- sub_date("2205", 10) % subtract 10 days from 22nd of May
"1205" % 12th of May
```

1.2. Running environment

During our laboratory work, we will use SWI-Prolog, an advanced implementation of the Prolog programming language known for its extensive libraries and robust debugging tools. To use SWI-Prolog, access it online. This allows you to write, test, and refine your Prolog code directly in your browser without any installation. This setup is ideal for quickly developing and iterating on complex logic-based applications.

2. Logic flow of the solution

The Prolog solution handles date-related operations, including converting date strings and day/month values, determining the day of the year from a given date, and managing leap years. The string_to_date function can either convert a date string (e.g., "0102" for 1st February) to day and month values or create a date string from given day and month values. The months' function lists month names used in various operations. The find_months_before function identifies all months preceding a specified month, and months_to_days convert these months into the number of days they contain. The date_month_to_month_start function calculates the starting day number of a given month by summing the days of all preceding months. The date_to_day_of_year function can either convert a given day and month to the corresponding day of the year or derive the day and month from a given day. The is_leap_year function checks if a year is a leap year based on divisibility rules. The month_days function provides the number of days each month, accounting for leap years for February. Finally, the add_date and sub_date functions adjust a date by adding or subtracting a specified number of days, respectively, converting between date formats and day-of-the-year representations as needed.

3. Main components of the code

his section highlights the primary predicates and their roles within the Prolog code for manipulating dates:

- **String to Date Conversion (**`string_to_date/3`**):** This predicate is critical for transforming a date string formatted as "DDMM" into separate day and month numerical components. It also performs the reverse operation, formatting numerical day and month values into a string. This functionality is key for processing dates for further operations.
- **Day of Year Calculation (**`date_to_day_of_year/3`**):** Calculates the day of the year for a given day and month. This calculation is essential for operations that require understanding a date's position within the year, such as calculating durations between dates or scheduling future events based on a date.
- **Date Arithmetic (**`add_date/2` **and** `sub_date/2`**):** These predicates handle adding or subtracting a specified number of days to or from a given date, effectively adjusting the date forward or backwards. This is particularly useful for planning and time-sensitive computations in various applications.
- **Leap Year Check (**`is_leap_year/1`**):** Determines if a year qualifies as a leap year, which is crucial for accurate date calculations involving the month of February.
- **Month Days Calculation (**`month_days/3`**):** Provides the number of days in any month, accounting for variations due to leap years. This predicate supports precise date arithmetic and year progression.
- **Utility Functions (**`e.g. months/1,find_months_before/2`**):** These helper functions offer foundational support such as providing a list of all months, calculating which months precede a specific month, and other basic functionalities that facilitate the primary operations of the data manipulation system.

4. Summary and challenges during the exercise

The main challenge during the exercise was understanding how Prolog works. Switching from imperative to declarative programming was also quite difficult, as we had previous experience with the first but not with the latter. In the beginning, we also thought that getting an inverse of a function would be simple and only require providing different parameters. Unfortunately, this was not always the case, and we sometimes needed to provide two different definitions for functions, each of which transformed one side or the other. Nevertheless, learning how to program in Prolog was an interesting experience, as Prolog differs greatly from languages we know.