

**CZ3005: Artificial Intelligence**

**2020 Fall Assignment 4:**

**Implementing a Talking Box with Prolog**

**Question 4:**

**Patient with a Sympathetic Doctor**

**Name: Royce Ang Jia Jie**

**Matriculation Number: U1840416D**

**Tutorial Group Number: TSP2**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**2020/2021 SEMESTER1**

Table of Contents

[Section 1. Introduction 3](#_Toc56737743)

[Section 2. Overview 3](#_Toc56737744)

[2.1 Introduction to KBS 3](#_Toc56737745)

[2.2 Overview of KBS 4](#_Toc56737746)

[2.3 Logic flow of Sympathetic Doctor Program 5](#_Toc56737747)

[5](#_Toc56737748)

[2.4 Further Considerations 6](#_Toc56737749)

[**2.4.1 Pain Level and its effect on the biasness of diagnosis** 6](#_Toc56737750)

[**2.4.2 Pain and Mood Level & their effect on the degree of sympathy** 6](#_Toc56737751)

[Section 3. Implementation and Explanation 7](#_Toc56737752)

[3.1 Code Explanation 7](#_Toc56737753)

[3.2 Logic Flow of Main Program 10](#_Toc56737754)

[3.3 Logic Flow for Diagnosis 12](#_Toc56737755)

[3.4 Heuristics/ Algorithm for conflicting diagnosis 14](#_Toc56737756)

[3.5 Knowledge Database 15](#_Toc56737757)

[3.6 Helper Functions 16](#_Toc56737758)

[3.7 Further Considerations (Error Handling) 16](#_Toc56737759)

[Section 4. Demonstration of Sympathetic Doctor Dialogue AI 16](#_Toc56737760)

[Section 5: Additional Implementation (GUI) with Google Speech Recognition Engine 18](#_Toc56737761)

[5.1 Motivation for GUI Implementation (Additional Feature) 18](#_Toc56737762)

[5.2 Overall System Architecture Diagram of the GUI Implementation 18](#_Toc56737763)

[5.3 Set up and Installation 18](#_Toc56737764)

[5.4 Implementation of speech recognition on GUI 19](#_Toc56737765)

[5.5 Screenshot of GUI Implementation 20](#_Toc56737766)

[5.6 Demonstration of Sympathetic Doctor Dialogue AI with GUI (YouTube Link) 20](#_Toc56737767)

[Works Cited 21](#_Toc56737768)

# **Section 1. Introduction**

Prolog is a logic programming language associated with Artificial Intelligence and Computational Linguistics.

It is interesting to note that unlike other programming languages, Prolog has its roots in first-order logic, a formal logic, used to represent a powerful Knowledge-Based System (KBS). As such, logic programmers can ascribe the efficiency of the knowledge-based representation to its generic representation without having to deal with technical concepts (time, events). Additionally, it is a universal language, allowing us to express anything that can be programmed.

In this assignment, we will:

(1) Implement a *Knowledge Base System Dialogue AI* to explore the concept of separation of *Knowledge* (essential representation of the world) from the *Inference Engine* (Set of inference rules) in KBS using Prolog.

(2) As part of an additional feature, we will be implementing a *GUI* to simplify and have an active interaction with the KBS, while observing how the KBS does the update in the background.

# **Section 2. Overview**

## **2.1 Introduction to KBS**

The KBS is designed with the following considerations:

1. The KBS must be able to ask user(patient) questions.
2. The user(patient) must be able to reply “yes” or “no” only.
3. The KBS must be able to ask the user(patient) 5 or more degree of pain level.
4. The KBS must be able to ask the user(patient) 5 or more levels of mood.
5. The KBS must be able to ask questions in an appropriate demeanour based on the user’s(patient) pain and mood level.
6. The KBS must have a total of 5 diseases.
7. For each disease, there must be at least 5 or more symptoms associated to it.
8. The KBS must be able to diagnose the patient’s disease.

Some additional considerations during the design of the KBS:

1. When a user(patient) replies a “yes” to any question, KBS must be able to *assert* the given answer as true.
2. The KBS is assumed to be a *thorough* AI. In which, the KBS should be able to do a *comprehensive analysis* of various symptoms should be done before making a diagnosis.
   1. KBS should be able to iterate through all symptoms.
   2. A good heuristic or algorithm should be implemented for diseases with overlapping symptoms.
3. The KBS must be able to answer the user(patient) in a sympathetic manner.
   1. A combination of pain level and mood level should output a type of gesture.
   2. Gesture chosen should be able to distinctively show an appropriate level of sympathy.

## **2.2 Overview of KBS**

To be able to separate the knowledge base from the inference engine, the KBS has the following 5 distinct “Databases”. To ensure that the symptoms of each disease/illness is as *realistic* as possible, I have referenced them from *NHS disease database* [1].

A picture containing graphical user interface

Description automatically generated

Figure 1 5 Distinct "Databases"

## **2.3 Logic flow of Sympathetic Doctor Program**

## A picture containing shape Description automatically generated

Figure 2 Sympathetic Doctor Logic Flow

Firstly, the Doctor will ask the patient if he/she is feeling any pain?

e.g. Doctor : “Hello! I am Doctor Box. Do you feel any pain?”

If the patient replies a “*No*”, his reply would implicitly assert that the patient is not having pain or “***pain free***”.

Otherwise, if the patient replies a “*Yes*”, this would initiate the iteration of the remaining list of 4 pain questions, until it is empty.

e.g. Doctor: “*Do you feel* ***mild pain***” (*no*)🡪 “ *Do you feel* ***moderate pain***” (*no*)🡪

Doctor: “*Do you feel* ***severe pain***” (*no*)🡪 “*Do you feel* ***overwhelmingly severe pain***”   
 (*no*)🡪 {*Empty List*}

Edge cases such as having an empty list is handled by implicitly asserting that the patient is not having pain or “***pain free***”.

The same procedure is repeated for asking *Patient’s Mood Level* (*Calm, Worried, Stressed, Fearful and Panic Stricken*).

The combination of pain level and mood level would determine how the Doctor should respond to the patient from one of the 9 varying types of gestures ( *Humorous, Attentive, Accommodating, Amiable, Very Attentive, Console, Comfort, Reassure and Companion*).

e.g. Overwhelming Pain + Panic Stricken 🡪 Companion Gesture

Doctor: “***Let us go through this together*** (*Gesture*). Do you have a cough? (*Symptom*)”

## **2.4 Further Considerations**

### **2.4.1 Pain Level and its effect on the biasness of diagnosis**

To simulate a real-life situation, where a doctor would use pain level to bias towards a specific type of diagnosis.

An additional feature is added such that pain level would also induce the biasness of diagnosis. This is implemented by **adding** *additional weights* (*heuristic*) to *diseases* associated to specific pain level; thereby, giving these disease(s) a heard start.

Note: This will be explained in the *implementation and explanation section*.

A picture containing shape

Description automatically generated

Figure 3 Weighted Disease(s) Associated to specific Pain Level

### **2.4.2 Pain and Mood Level & their effect on the degree of sympathy**

Another challenge faced was the designing of gestures. A great emphasis is placed on good patient-physician communication skill to ensure the authenticity of the dialogue with the Doctor [2]. This is done by combining both non-verbal (e.g. *emoticons*) and verbal cues (e.g*. filler words*) for a single gesture.

e.g. *gesture(humorous) :- knowledgeable(A), emoticons(B), kidding(C), filler words(D)*

Note the mood level in the following order:

***Calm*** *<* ***Worried*** *<* ***Stressed*** *<* ***Fearful*** *<* ***Panic Stricken***

The degree of seriousness in gestures are as follow:

***Humorous*** *<* ***Amiable*** *<* ***Attentive*** *<* ***Very Attentive*** *<* ***Accommodating*** *<* ***Console*** *<* ***Comfort*** *<* ***Reassure*** *<* ***Companion***

Additionally, because of the extreme case, where a patient may experience (1) ***Overwhelmingly Severe Pain*** and (2) ***Calm Mood***, it would be inappropriate to have a gesture of ***Humorous***, but a more (in-between) serious degree of gesture should be picked instead, for instance, the gesture of ***Comfort****.*

Therefore, when a patient experiences (1) ***Overwhelmingly Severe Pain*** *and (2)* ***Panic Stricken*** *🡪* ***Companion*** *Gesture* should beexpected (since the degree of seriousness in the list of gestures is the highest), thereby, providing justified comfort to the patient.

# **Section 3. Implementation and Explanation**

## **3.1 Code Explanation**

**Predicates/Variables used:** (\*More details in source code\*)

1. pain\_questions(L). – Provides a list of question for the patient to access level of pain.
2. knowledgable(L). – Provides a list of knowledgeable gestures a doctor would say.
3. emoticons(L). – Provides a list of non-verbal cues/gestures for a doctor to say.
4. kidding(L). – Provides a list of joking gesture for the doctor to say.
5. filler\_words(L). – Provides a list of thinking gestures for the doctor to say.
6. companion(L). – Provides a list of hearty gestures for the doctor to say.
7. relax(L). – Provides a list of reassuring gestures for the doctor to say.
8. inspiring\_quote(L). – Provides a list of inspiring gestures for the doctor to say.
9. attentive(L). – Provides a list of attentive gestures for the doctor to say.
10. count\_acne(X). – “Heuristic” value for acne.
11. count\_flu(X). – “Heuristic” value for flu.
12. count\_allergy(X). – “Heuristic” value for allergy.
13. count\_covid\_19(X). – “Heuristic” value for covid-19.
14. count\_heart\_disease(X). – “Heuristic” value for heart disease.
15. count\_high\_blood\_sugar(X). – “Heuristic” value for high blood sugar.
16. count\_cancer(X). – “Heuristic” value for cancer.

**Rules that govern the mood of the patient:**

1. confirm\_mood(calm) :- assert(mood(calm)).
2. confirm\_mood(worried) :- assert(mood(worried)).
3. confirm\_mood(stressed) :- assert(mood(stressed)).
4. confirm\_mood(fearful) :- assert(mood(fearful)).
5. confirm\_mood(panic\_stricken) :- assert(mood(panic\_stricken)).

**Rules that govern all gestures the doctor can have determined by the patient’s pain level and mood level.**

1. all\_reactions(L) :- pain(pain\_free),mood(calm), assertz(gesture(humorous)), all\_gesture(L).
2. all\_reactions(L) :- pain(mild\_pain),mood(calm), assertz(gesture(humorous)), all\_gesture(L).
3. all\_reactions(L) :- pain(moderate\_pain),mood(calm), assertz(gesture(humorous)), all\_gesture(L).
4. all\_reactions(L) :- pain(severe\_pain),mood(calm), assertz(gesture(attentive)), all\_gesture(L).
5. all\_reactions(L) :- pain(overwhelming\_pain),mood(calm), assertz(gesture(accommodating)), all\_gesture(L).
6. all\_reactions(L) :- pain(pain\_free),mood(worried), assertz(gesture(amiable)), all\_gesture(L).
7. all\_reactions(L) :- pain(mild\_pain),mood(worried), assertz(gesture(amiable)), all\_gesture(L).
8. all\_reactions(L) :- pain(moderate\_pain),mood(worried), assertz(gesture(amiable)), all\_gesture(L).
9. all\_reactions(L) :- pain(severe\_pain),mood(worried), assertz(gesture(very\_attentive)), all\_gesture(L).
10. all\_reactions(L) :- pain(overwhelming\_pain),mood(worried), assertz(gesture(console)), all\_gesture(L).
11. all\_reactions(L) :- pain(pain\_free),mood(stressed), assertz(gesture(comfort)), all\_gesture(L).
12. all\_reactions(L) :- pain(mild\_pain),mood(stressed), assertz(gesture(comfort)), all\_gesture(L).
13. all\_reactions(L) :- pain(moderate\_pain),mood(stressed), assertz(gesture(comfort)), all\_gesture(L).
14. all\_reactions(L) :- pain(severe\_pain),mood(stressed), assertz(gesture(comfort)), all\_gesture(L).
15. all\_reactions(L) :- pain(overwhelming\_pain),mood(stressed), assertz(gesture(comfort)), all\_gesture(L).
16. all\_reactions(L) :- pain(pain\_free),mood(fearful), assertz(gesture(reassure)), all\_gesture(L).
17. all\_reactions(L) :- pain(mild\_pain),mood(fearful), assertz(gesture(reassure)), all\_gesture(L).
18. all\_reactions(L) :- pain(moderate\_pain),mood(fearful), assertz(gesture(reassure)), all\_gesture(L).
19. all\_reactions(L) :- pain(severe\_pain),mood(fearful), assertz(gesture(reassure)), all\_gesture(L).
20. all\_reactions(L) :- pain(overwhelming\_pain),mood(fearful), assertz(gesture(reassure)), all\_gesture(L).
21. all\_reactions(L) :- pain(pain\_free),mood(panic\_stricken), assertz(gesture(companion)), all\_gesture(L).
22. all\_reactions(L) :- pain(mild\_pain),mood(panic\_stricken), assertz(gesture(companion)), all\_gesture(L).
23. all\_reactions(L) :- pain(moderate\_pain),mood(panic\_stricken), assertz(gesture(companion)), all\_gesture(L).
24. all\_reactions(L) :- pain(severe\_pain),mood(panic\_stricken), assertz(gesture(companion)), all\_gesture(L).
25. all\_reactions(L) :- pain(overwhelming\_pain),mood(panic\_stricken), assertz(gesture(companion)), all\_gesture(L).

**Rules that govern “heuristic” to resolve the issue of diseases with overlapping symptoms.**

For instance, rule 1 is an overlapping symptom with acne and cancer, therefore, the increment weight for both diseases is 1. However, in rule 2, because “whiteheads” is a strong indicator of acne, the weight increment for acne is 2.

1. incr\_lump :- (add\_count\_acne, add\_count\_cancer).
2. incr\_whiteheads :- add\_count\_acne, add\_count\_acne.
3. incr\_blackheads :- add\_count\_acne, add\_count\_acne.
4. \*view more in the source code\*

**Rules that diagnose the patient based on the “heuristic”. Highest value will serve as the diagnosis.**

1. diagnose(acne) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

A >= B, A >= C, A >= D, A >= E, A >= F, A >= G.

1. diagnose(flu) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

B >= A, B >= C, B >= D, B >= E, B >= F, B >= G.

1. diagnose(allergy) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

C >= B, C >= A, C >= D, C >= E, C >= F, C >= G.

1. diagnose(covid\_19) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

D >= B, D >= C, D >= A, D >= E, D >= F, D >= G.

1. diagnose(heart\_disease) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

E >= B, E >= C, E >= D, E >= A, E >= F, E >= G.

1. diagnose(high\_blood\_sugar) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

F >= B, F >= C, F >= D, F >= E, F >= A, F >= G.

1. diagnose(cancer) :- count\_acne(A), count\_flu(B), count\_allergy(C), count\_covid\_19(D), count\_heart\_disease(E), count\_high\_blood\_sugar(F), count\_cancer(G),

G >= B, G >= C, G >= D, G >= E, G >= F, G >= A.

**Predicates that asserts symptom/pain and increments the appropriate counter.**

1. merge(lump) :- (assert(has(lump)), incr\_lump).
2. merge(whiteheads) :- (assert(has(whiteheads)), incr\_whiteheads).
3. merge(blackheads) :- (assert(has(blackheads)), incr\_blackheads).
4. *\*view more in source code under merge\**
5. confirm\_pain(mild\_pain) :- assert(pain(mild\_pain)), add\_count\_acne, add\_count\_allergy.
6. confirm\_pain(moderate\_pain) :- assert(pain(moderate\_pain)), add\_count\_high\_blood\_sugar, add\_count\_covid\_19.
7. *\*view more in source code under confirm\_pain\**

## **3.2 Logic Flow of Main Program**

Step 1: Initialization of Main Program via *ask(0).*

Text

Description automatically generated

Step 2: Selects pain question from pain question ‘*Database*’ and return back to *ask\_repeat* program.

Text

Description automatically generated

Step 3: When pain level has been selected, proceed to (1) *confirm\_pain(pain)* level & (2) *query mood.*

Text

Description automatically generated

Step 3.1: Confirm pain level by (1) *asserting the pain level* and (2) *increasing the total initial heuristics/counts* towards the associated disease(s). As explained previously in ***2.4.1*** , this is an additional consideration/feature to provide a head start for some diseases due to the extra initial weight provided, thereby, providing a more realistic approach towards diagnosing the disease.

Text

Description automatically generated

Step 4: Selects mood question from the list of mood questions ‘*Database*’ and return back to *ask\_mood\_repeat* program.

Text

Description automatically generated

Step 5: When mood level has been selected, proceed to (1) *confirm\_mood(mood)* and (2) *query\_symptoms(X)* program.

Text

Description automatically generated

Step 5.1: Confirms the mood by asserting the mood level provided by the patient.

Text

Description automatically generated

Step 6.1: Query for a random symptom and with the appropriate gesture from the Doctor, ask the patient sympathetically.

Text

Description automatically generated

Step 6.2: Selects an appropriate Gesture before returning to *Step 6.1*  to ask the patient sympathetically.

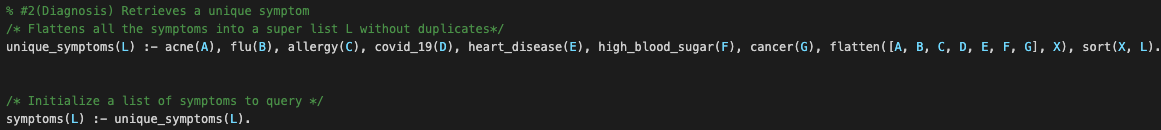


## **3.3 Logic Flow for Diagnosis**

Step 1: Query for random symptom



Step 2: Selects a unique symptom for Step 1



Step 3: If patient answers “*Yes*” to a symptom, (1) *assert that symptom* and (2) *proceed to merge(symptom) program*.

Text

Description automatically generated

Step 4: Assert the appropriate symptom and increment the symptom accordingly.(*This is a snippet*)

Text

Description automatically generated

Step 5: Increments Disease Counter (*This is only a snippet, please refer to the code for the full implementation*)

Text

Description automatically generated

Step 6: Increments “heuristic”/count\_value for a specific disease.

(*This is only a snippet, please refer to the code for the full implementation*)

Text

Description automatically generated

Step 7: When symptom list is empty, proceed to diagnose program. Disease with the *highest heuristic* will be considered for the diagnosis of the patient.

Text

Description automatically generated

## **3.4 Heuristics/ Algorithm for conflicting diagnosis**

One of the problems faced is the possibility of 2 diseases with overlapping symptom, for instance, count\_flu = X and count\_cancer = X. However, patient is diagnose with flu instead of cancer mainly due to the order. As such, to resolve this problem, I’ve implemented a ***weighted/heuristic system*** whereby the count will be incremented more if the symptom is a stronger indicator for that specific disease.

The heuristic is governed by the following algorithm:

(1) For disease with non-unique symptoms, the increment weight is 1.

(2) For disease with total unique symptoms > 2, the increment weight is 2.

(3) For disease with total unique symptoms <=2, the increment weight is 3.

*(More explanation on implementation can be found in source code under Heruistics section)*

Text

Description automatically generated

Figure 4 Unique symptoms == Stronger Indicator (Non-Highlighted & Non-Underlined)

## **3.5 Knowledge Database**

Disease Database

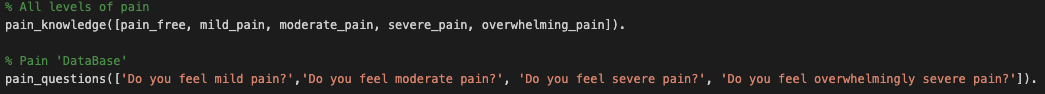


Symptoms Database

Text

Description automatically generated

Pain Database (Note: ‘pain\_free’ question is asked in ask(0) program of “Do you feel any pain?”)



Mood Database



Gestures Database

A screen shot of a computer

Description automatically generated

## **3.6 Helper Functions**

Text

Description automatically generated

## **3.7 Further Considerations (Error Handling)**

Rules governing pain query empty list (error handling).

**Text

Description automatically generated**

Rules governing mood query empty list (error handling).

Text

Description automatically generated

# **Section 4. Demonstration of Sympathetic Doctor Dialogue AI**

Case 1: Cancer disease with (1) **mild pain** and (2) **calm mood**. (Notice the gesture)

Text

Description automatically generated

Case 2: Cancer disease with (1) **mild pain** and (2) **panic stricken mood**. (Notice the difference in gesture from case 1)

Text

Description automatically generated

Case 3: Different level of pain and different diagnose result.

Text

Description automatically generated

Case 4: Handle Empty List Error; when all pain level and mood level are answered with “no”.

Text

Description automatically generated

# **Section 5: Additional Implementation (GUI) with Google Speech Recognition Engine**

## **5.1 Motivation for GUI Implementation (Additional Feature)**

While it is entirely feasible to have Prolog KBS to be the forefront of the application, it still presents its form of inconvenience to users. For instance, replying to KBS requires the user to spell both "no" and "yes" correctly. And as humans, we tend to err easily, and this could prove to be catastrophic if the KBS is used to run on a larger scale. Misspelling “no” and “yes” would cause a system crash in the KBS.

As such, to resolve this issue. Having an interactable GUI would serve as a form of layered abstraction, thereby, hiding the internal logic of the KBS from the user. The chances of a system crash would be reduced significantly.

## **5.2 Overall System Architecture Diagram of the GUI Implementation**

A picture containing shape

Description automatically generated

Figure 5 System Architecture Diagram of Implemented GUI

## **5.3 Set up and Installation**

1. Ensure that you have Python and SWI-Prolog installed on your computer.
   1. Python 2.7 or 3.4 and higher.
   2. SWI-Prolog 7.2.x and higher.
   3. Both must have similar bit architecture. (E.g both 64 bit).
2. Install PySwip using the command: pip install pyswip
3. Install PySimpleGUI using the command: pip
4. Install PySimpleGUI Pyttsx3 using the command: pip install pyttsx3
   1. Additionally, for windows, install Pypiwin32 using the command: pip install pypiwin32
5. Install SpeechRecognition using the command: pip install SpeechRecognition
   1. Install pyaudio using the command: pip install pyaudio  
      For python 3.7 and above, download the appropriate file from https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyaudio
   2. Example Python 3.8, 64 bit -> download PyAudio-0.2.11-cp38-cp38-
   3. win\_amd64
   4. Go to the directory where the file is and install it using the command: pip install [file\_name].whl

## **5.4 Implementation of speech recognition on GUI**

(*Full implementation can be found in source code)*

Here, I will only highlight the class I’ve used to instantiate a speech recognition engine used in this GUI. Also note that, running a speech recognition is thread-intensive. As such, I’ve only implemented the speech recognition to instantiate twice. Once at the beginning of the program, “Hello! I am Doctor Box. Do you feel any pain?”, and the other at the end of the program, when the Doctor diagnoses, “I diagnose that you have a….”.

Text

Description automatically generated

Figure 6 Speech Recognition Class

## **5.5 Screenshot of GUI Implementation**

Notice that a “Yes” and “No” buttons are presentfor the patient to interact with, without having to worry about the internal logic of the KBS.

**Graphical user interface, text

Description automatically generated**

## **5.6 Demonstration of Sympathetic Doctor Dialogue AI with GUI (YouTube Link)**

The demonstration of the Sympathetic Doctor GUI can be found in the zipped folder or via this YouTube link:

**Section 6. Conclusion**

# **Works Cited**

|  |  |
| --- | --- |
| [1] | A. K. A. C. Piyush Ranjan, "How can Doctors Improve thier Communication Skills?," *NCBI,* vol. 9, no. 3, pp. 01-04, 2015. |
| [2] | NHS, "NHS inform," [Online]. Available: https://www.nhsinform.scot/illnesses-and-conditions/immune-system/allergies. [Accessed 19 November 2020]. |