Motor de encadenamientos.

García García Jonathan Eduardo
jgarciag1404@alumno.ipn.mx

González Santiesteban Santiago
sgonzalezs1400@alumno.ipn.mx

Jimenez Angeles Daniel Antonio
djimeneza1400@alumno.ipn.mx

Abstracto—Este documento es el reporte técnico, explicativo de la estructura interna de este motor de encadenamientos.

Palabras clave—Encadenamiento, inferencia, sistema experto, toma de decisiones

I. Introducción

Un motor de inferencia es el encargado de manejar el proceso de selección, decisión, interpretación y aplicación del comportamiento que refleja el razonamiento. Procesa e interpreta reglas que se encargan de resolver un problema de decisión. En la lógica clásica, es posible deducirse mediante el empleo de reglas, si su premisa es cierta, también lo será su conclusión.

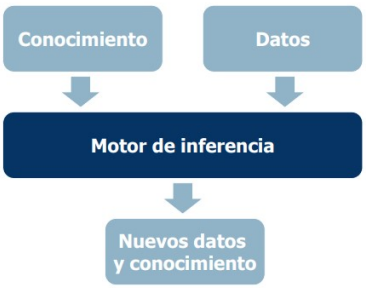
II. Marco teórico

A. Sistema experto

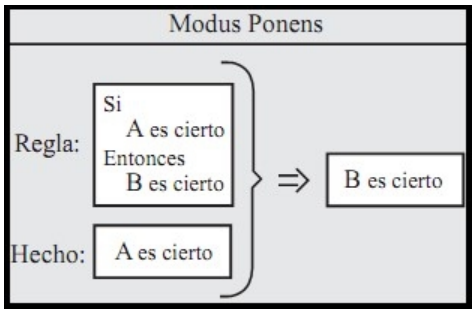
Se puede decir que los Sistemas Expertos son el primer resultado operacional de la Inteligencia artificial, pues logran resolver problemas a través del conocimiento y raciocinio de igual forma que lo hace el experto humano. Un Sistema Experto (SE), es básicamente un programa de computadora basado en conocimientos y raciocinio que lleva a cabo tareas que generalmente sólo realiza un experto humano9; es decir, es un programa que imita el comportamiento humano en el sentido de que utiliza la información que le es proporcionada para poder dar una opinión sobre un tema en especial. Otros autores lo definen como sigue:
un Sistema Experto es un programa de computadora interactivo que contiene la experiencia, conocimiento y habilidad propios de una persona o grupos de personas especialistas en un área particular del conocimiento humano, de manera que permitan resolver problemas específicos de ése área de manera inteligente y satisfactoria10. La tarea principal de un SE es tratar de aconsejar al usuario. Los usuarios que introducen la información al SE son en realidad los expertos humanos, y tratan a su vez de estructurar los conocimientos que poseen para ponerlos entonces a disposición del sistema. Los SE son útiles para resolver problemas que se basan en conocimiento.

B. Modus Ponens

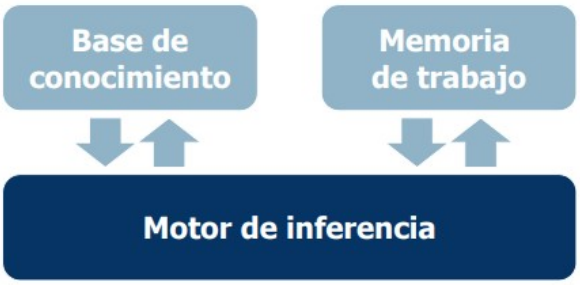
Es quizá la regla de inferencia más comúnmente utilizada. Se utiliza para obtener conclusiones simples, en ella se analiza la premisa de la regla, y si es cierta, la conclusión entra a formar parte del conocimiento. Como ilustración supóngase que se tiene la regla, “Si A es cierto, entonces



B es cierto”, y que se sabe además que “A es cierto”. La regla Modus Ponens, concluye que “B es cierto”.



C. Arquitectura de un sistema experto



III. Desarrollo

El motor de inferencia están escritos en C# ”C Sharp” debido a que este lenguaje de programación implementa los pilares de POO.

- Abstracción
  Permite diseñar cada uno de los modulos de un microprocesador a nivel de objetos.

- Polimorfismo
  Permite sobrecargar métodos teniendo un único patrón en conmún.
- Encapsulamiento
  Protege los datos almacenados en la memoria limitando la lectura ó escritura de los mismos a menos que seán autorizados explicitamente.
- Herencia
  Permite compartir fragmentos de código entre clases heredando métodos y atributos evitando la redundacia en el código.

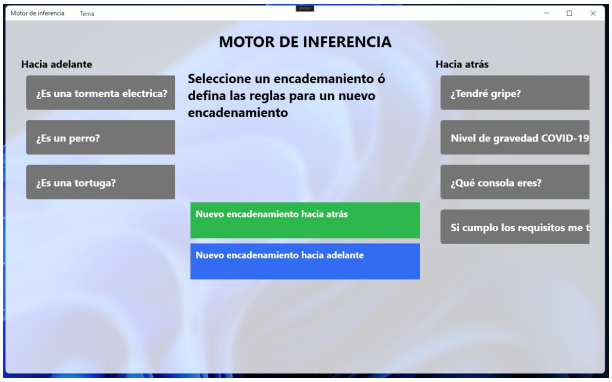
A. Generalidades

El programa permite definir y ejecutar las estrucutras de Encadenamiento hacia adelante y hacia atras Entre los módulos implementados para cumplir con el objetivo se encuentran:

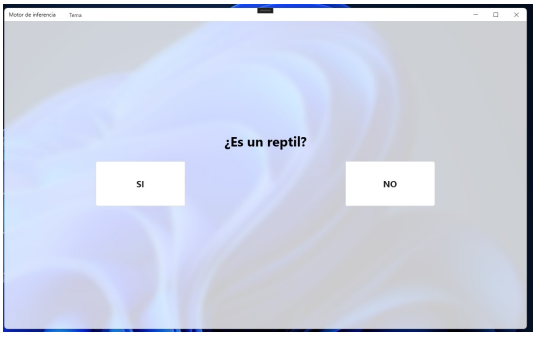
- Definición y edición de encadenamiento hacia adelante
- Definición y edición de encadenamiento hacia atrás.
- Base de datos de sqlite (donde se almacena la base de conocimiento)
- Interfaz de ejecución de encadenamiento hacia adelante
- Interfaz de ejecución de encadenamiento hacia atras

IV. Proceso de Ejecución

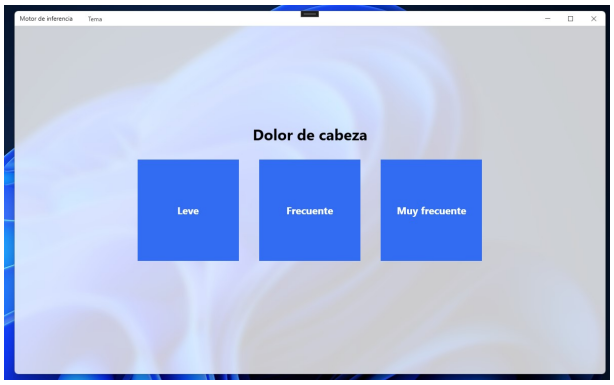
Seleccionar el encadenamiento que se desea ejecutar:



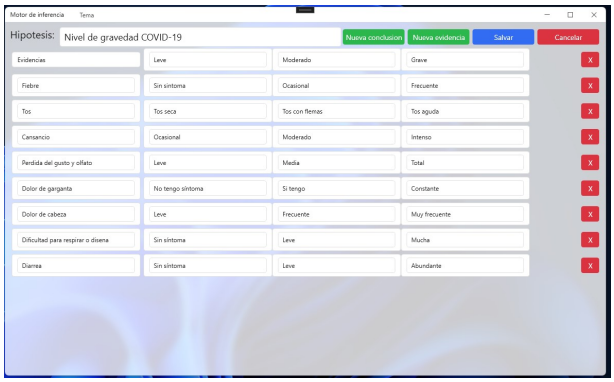
Encadenamiento hacia adelante:



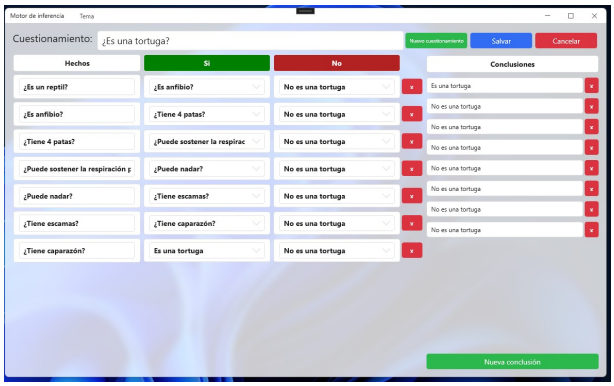
Encadenamiento hacia atrás: Definición y edición de



encadenamiento hacia atrás:



Definición y edición de encadenamiento hacia adelante:



V. Código de las clases principales

A. IQuestion

IQuestion

```
    public interface IQuestion
    {
        public bool Answer { get; set; }
        public string Question { get; set; }
        public void Ask(MainView mainView);
    }
```

B. IEncadenamiento

IEncadenamiento

```
    public interface IEncadenamiento
    {
        [JsonIgnore]
        public string Title { get; }
        [JsonIgnore]
        public string Type { get; }
    }
```

C. Engine

Engine

```
        public static class Engine
        {
            public static void Run(MainView mainView, IEncadenamiento encadenamiento)
            {
                switch (encadenamiento)
                {
                    case HaciaAtras atras:
                        Run(mainView, atras.Hipotesis);
                        return;
                    case HaciaAdelante adelante:
                        Run(mainView, adelante.Hecho);
                        return;
                }
            }
            public static void Run(MainView mainView, Atras.Hipotesis hipotesis)
            {
                mainView.Content = new StartupView(new StartupViewModel(mainView, hipotesis));
            }
            public static void Run(MainView mainView, Adelante.Hecho hecho)
            {
                mainView.Content = new StartupView(new StartupViewModel(mainView, hecho));
            }
        }
```

D. Conclusion

Conclusion

```
public class Conclusion : IGuid, IComparable<Conclusion>, IEquatable<Conclusion>, IBranch
{
    public string Descripcion { get; set; }
    [PrimaryKey]
    public Guid Guid { get; set; }
    public bool Side { get; set; }
    public Guid ParentId { get; set; }
    public Conclusion()
    {

    }
    public Conclusion(string descripcion)
    {
        Descripcion = descripcion;
    }
    public static Conclusion New(string descripcion) => new Conclusion(descripcion);

    public void Run(MainView window)
    {
        var view = new ConclusionView();
        var model = new ConclusionViewModel(window, this);
        view.DataContext = model;
        window.Content = view;
    }

    public void Delete()
    {
        App.SqLite.Delete(this);
    }
    public void Save()
    {
        App.SqLite.InsertOrReplace(this);
    }

    public void Save(Guid Guid, bool answer)
    {
        if (ParentId != Guid.Empty)
        {
            this.Guid = Guid.NewGuid();
        }
        ParentId = Guid;
        Side = answer;
        Save();
    }

    public void Load()
    {
        //nada que cargar :)
    }

    public bool Equals(Conclusion other)
    {
        return other?.Guid == this.Guid;
    }

    public override int GetHashCode()
    {
        var a = this.Guid.GetHashCode();
        return a;
    }

    public override string ToString() => Descripcion;

    public int CompareTo([AllowNull] Conclusion other) => this.Guid.CompareTo(other?.Guid ?? Guid.
        Empty);
}
```

E. Hipotesis

Hipotesis

```
1   public class Hipotesis : IQuestion, IBranch, IGuid
2   {
3       [PrimaryKey]
4       public Guid Guid { get; set; }
5       [Ignore]
6       public bool Answer { get; set; }
7       [Ignore]
8       public string Descripcion => Question;
9       public string Question { get; set; }
10      [Ignore]
11      public IBranch Verdadero { get; set; }
12      [Ignore]
13      public IBranch Falso { get; set; }
14      public bool Side { get; set; }
15      public Guid ParentId { get; set; }
16
17      public new static Hipotesis New(string descripcion) => new Hipotesis(descripcion);
18      public Hipotesis(string question)
19      {
20          Question = question;
21      }
22
23      public Hipotesis()
24      {
25
26      }
27
28      public Hipotesis Set(IBranch verdadero, IBranch falso)
29      {
30          Verdadero = verdadero;
31          Falso = falso;
32          return this;
33      }
34
35      public void Run(MainView main) => Ask(main);
36      public void Ask(MainView main)
37      {
38          var view = new QuestionView();
39          var model = new QuestionViewModel(main, this);
40          view.DataContext = model;
41          main.Content = view;
42      }
43
44      public void Delete()
45      {
46          App.SqLite.Delete(this);
47          this.Verdadero.Delete();
48          this.Falso.Delete();
49      }
50      public void Save() => Save(Guid.Empty, false);
51
52      public void Save(Guid pGuid, bool answer)
53      {
54          ParentId = pGuid;
55          Side = answer;
56          App.SqLite.InsertOrReplace(this);
57          this.Verdadero.Save(Guid, true);
58          this.Falso.Save(Guid, false);
59      }
60
61      public void Load()
62      {
63
64          this.Verdadero=(IBranch)
65              App.SqLite.Table<Hipotesis>().FirstOrDefault(x => x.ParentId == this.Guid && x.Side)
66              ?? App.SqLite.Table<Conclusion>().First(x => x.ParentId == this.Guid && x.Side);
67
68          this.Falso = (IBranch)
69                          App.SqLite.Table<Hipotesis>().FirstOrDefault(x => x.ParentId == this.Guid
                                && !x.Side)
70                          ?? App.SqLite.Table<Conclusion>().First(x => x.ParentId == this.Guid && !x
                                .Side);
71          this.Verdadero.Load();
72          this.Falso.Load();
73
74      }
75  }
```