Joshua

LosAngeles971, 2021

Joshua

> I am not afraid of how much a machine can look like a human being, but how much a human being can be assimilated to a complex machine.
>
> — @Bluepulsar971

Joshua is a software. It was born from an inspiration coming from reading a book [1], and its purpose is solving generics problems using a combination of **knowledge** and **state**. The knowledge consists of **events** and relationships between events. An event is *something* that can occur under certain circumstances, for example an event may be "the farmer reaches the est bank of the river", and it can occur if the farmer is on the ovest bank of the river. A relationship ties two events under the principle of **cause-effect**, thus the occurrence of an event can cause the occurrence of other events. Finally, the state is a set of **variables** which can determine the conditions that allow an event to happen, as well as the occurrence of every event can change the state. How can Joshua solve a problem? Given a base knowledge, a problem can be described as the combination of an initial state and an event, and finding the solution means answering the question: **is there at leat one chain of events that eventually cause my event?**.

[1] ??? - ???

1. Introduction

TBD

2. The farmer, the wolf, the goat and the cabbage

> Once upon a time a farmer went to a market and purchased a wolf, a goat, and a cabbage. On his way home, the farmer came to the bank of a river and rented a boat. But crossing the river by boat, the farmer could carry only himself and a single one of his purchases: the wolf, the goat, or the cabbage. If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage. The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact. How did he do it?
>
> — https://en.wikipedia.org/wiki/Wolf, _goat_and_cabbage_problem

Joshua describes a problem as the combination of an initial **state** and a desired **effect**. Finding the solution of a problem means answering the question: is there at least one **chain of concatenated events** that eventually cause the desired effect?**

An event is something like a phisycal phenomena or an action, that can occur under certain *conditions*. For example, "The farmer brings the cabbage on the bank B of the river" can occur if both farmer and cabbage are on the bank A of the river. Conditions come form a **state**.

The state is a set of **variables** that over the time determine the **conditions** that allow events to occur, as well as the occurrence of events change the state.

Eventually, the definition of an event includes:

- a unique description
- an *optional* list of conditions
- an *optional* list of assignments
- an *optional* list of effects

Techically, an **effect** is just an event. Althought, an effect assumes a **cause-effect relationship** with another event.

Joshua recognizes three types of cause-effect relationships depending on their **weight**:

- a weight of 0 means that the cause **never** triggers the effect
- a weight of 1 means that the cause **always** triggers the effect
- any weight into (0,1) means that the cause *may* trigger the effect

You can see the weight as the probability that the cause can trigger the effect.

For example: the event "The farmer brings the cabbage on the bank B of the river" *may* cause the effect "The farmer, the wolf, the goat and the cabbage are on the bank B of the river", therefore the

weight of such relationship must be into the interval of (0,1).

You can see the weight as the probability that a casue-effect relationship can occur, though when the weight is into the interval (0,1), it is to necessary to specify a precise probability.

Back to the concept of state and thinking of Farmer's problem, we may define 8 variables in order to determine, over the time, if the characters are on the river's bank A (value 1) or not (value 0):

- FarmerA = 1
- WolfA = 1
- GoatA = 1
- CabbageA = 1
- FarmerB = 0
- WolfB = 0
- GoatB = 0
- CabbageB = 0

Given the defined variable, below a complete definition of one event.

Table 1. The definition of an event

Event	"The farmer brings the cabbage on the bank B of the river"
Conditions	FarmerA == 1 CabbageA == 1 WolfA < 1) && (GoatA == 1
	WolfA == 1) && (GoatA < 1
Effect	"The farmer, the wolf, the goat and the cabbage are on the bank B of the river" with weight 0.5

Assignments	`FarmerB = 1`
	`CabbageB = 1`
	`FarmerA = 0`
	`CabbageA = 0`

The **assignments** express the consequences of an event's occurrence. For example, if the event "The farmer brings the cabbage on the bank B of the river" happened, then state must change accordingly.

NOTE

A set of well known events and cause-effect relationships forms the base knowledge used by Joshua to solve a **problem**. Even if solving problems may increase the knowledge, Joshua is not able (or not deisgned to) to build up the knonledge from scratch, since it requires the ability to observe and understand the occurrence of not known events, in order to classify them and deriving new cause-effect relationships.

Starting from a base knowledge, solving the Farmer's problem means understanding what chains

of concatenated cause-effect relationships brings to the final effect "The farmer, the wolf, the goat and the cabbage are on the bank B of the river".

NOTE	For Joshua, **solving a problem** means understanding if a desired event can occur alone or as the final effect of a chain of concatenated cause-effect relationships.

Being able to describe a problem is everything, nevertheless it can be tricky, since the description has to be limited to declare the facts (events, relationships and state), **without falling into the desire of anticipating the reasoning**.

Describing a problem involves few important steps:

1. Identifying the event that represent the solution
2. Identifying all others relevant events
3. Identifying the cause-effect relationships between events
4. Identifying conditions and assignments for each event

Here the events of the Farmer's problem:

- The farmer, the wolf, the goat and the cabbage are on the bank B of the river (**solution**)
- The farmer brings the cabbage on the bank B of the river
- The farmer brings the cabbage on the bank A of the river
- The farmer brings the goat on the bank B of the river
- The farmer brings the goat on the bank A of the river
- The farmer brings the wolf on the bank B of the river
- The farmer brings the wolf on the bank A of the river
- The farmer goes to the bank A of the river
- The farmer goes to the bank B of the river

All events, *except the solution*, are the actions the Farmer can do, and all of them may cause the solution. Why all? The answer can be tricky, from the perspective of not anticipating any reasoning to Joshua. Indeed one may think that the event/action "The farmer goes to the bank A of the river" cannot be the cause of the solution, but that's true only if the solution require strictly one **one** event/action (that is a chain of only one cause-effect relationships).

Since the Farmer may require (we know he must require) a sequence of actions to solve the problem, then each of his action can be a step toward the solution.

Below a complete definition of events and relationships of the Farmer's problem, in the form (YAML file [2]) used by Joshua to define a base knowledge.

```
  knowlegde:
    - id: "The farmer, the wolf, the goat and the cabbage are on the bank B of the
river"
      conditions:
```

```
        - FarmerB == 1
        - WolfB == 1
        - GoatB == 1
        - CabbageB == 1

    - id: The farmer brings the cabbage on the bank B of the river
      conditions:
        - FarmerA == 1
        - CabbageA == 1
        - ((WolfA < 1) && (GoatA == 1)) || ((WolfA == 1) && (GoatA < 1))
      effects:
        - weight: 0.5
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerB = 1
        - CabbageB = 1
        - FarmerA = 0
        - CabbageA = 0

    - id: "The farmer brings the cabbage on the bank A of the river"
      conditions:
        - FarmerB == 1
        - CabbageB == 1
        - ((WolfB < 1) && (GoatB == 1)) || ((WolfB == 1) && (GoatB < 1))
      effects:
        - weight: 0.1
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerA = 1
        - CabbageA = 1
        - FarmerB = 0
        - CabbageB = 0

    - id: "The farmer brings the goat on the bank B of the river"
      conditions:
        - FarmerA == 1
        - GoatA == 1
      effects:
        - weight: 0.5
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerB = 1
        - GoatB = 1
        - FarmerA = 0
        - GoatA = 0

    - id: "The farmer brings the goat on the bank A of the river"
      conditions:
```

```
        - FarmerB == 1
        - GoatB == 1
      effects:
        - weight: 0.1
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerA = 1
        - GoatA = 1
        - FarmerB = 0
        - GoatB = 0

    - id: "The farmer brings the wolf on the bank B of the river"
      conditions:
        - FarmerA == 1
        - WolfA == 1
        - ((CabbageA < 1) && (GoatA == 1)) || ((CabbageA == 1) && (GoatA < 1))
      effects:
        - weight: 0.5
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerB = 1
        - WolfB = 1
        - FarmerA = 0
        - WolfA = 0

    - id: "The farmer brings the wolf on the bank A of the river"
      conditions:
        - FarmerB == 1
        - WolfB == 1
        - ((CabbageB < 1) && (GoatB == 1)) || (CabbageB == 1) && (GoatB < 1))
      effects:
        - weight: 0.1
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerA = 1
        - WolfA = 1
        - FarmerB = 0
        - WolfB = 0

    - id: "The farmer goes to the bank A of the river"
      conditions:
        - FarmerB == 1
        - WolfB == 1 || CabbageA == 1 || GoatA == 1
        - CabbageB == 1 || WolfA == 1 || GoatA == 1
        - WolfA == 1 || CabbageA == 1 || GoatA == 1
      effects:
        - weight: 0.3
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
```

```
the river"
      assignments:
        - FarmerA = 1
        - FarmerB = 0

    - id: "The farmer goes to the bank B of the river"
      conditions:
        - FarmerA == 1
        - WolfA == 1 || CabbageB == 1 || GoatB == 1
        - CabbageA == 1 || WolfB == 1 || GoatB == 1
        - WolfB == 1 || CabbageB == 1 || GoatB == 1
      effects:
        - weight: 0.5
          event: "The farmer, the wolf, the goat and the cabbage are on the bank B of
the river"
      assignments:
        - FarmerB = 1
        - FarmerA = 0
```

All conditions of the events only arise from the problem's facts. For example the event "The farmer brings the cabbage on the bank B of the river" can happen only if:

- Farmer and cabbage are on the bank A of the river (obvious consideration)
- Wolf and goat are not alone in the same place (the wolf would eat the goat)

Likewise, the assignments are the simple adaptions of the state.

NOTE	The proposed knowledge for the Farmer's problem is based on the idea "if you do not know what to do, any move can lead to the solution", Such approach is a **brutal force attack** to the problem, and a natural consequence of having many events that **may** cause the solution.

Joshua requires 24 cycles to solve the problem, most of them are foolish attempts.

```
joshua.exe solve -k ..\resources\k_contadino.yml -p ..\resources\p_contadino.yml
Max cycles:  100
State contains:  8  variables
        Outcome|   true|
         Cycles|     24|
   Queue's size|     31|
```

Starting from the desired effect, Joshua identifies all possible chain of concatenated events that may cause the solution. Joshua tries each chain, until the solution is reached.

Table 2. All attempts made by Joshua to solve the Farmer's problem.

Step	Event	Outcome	State changed?	Effects
0	The farmer brings the cabbage on the bank B of the river	No conditions	False	None
1	The farmer brings the goat on the bank B of the river	It happened	True	None
2	The farmer brings the wolf on the bank B of the river	No conditions	False	None
3	The farmer goes to the bank B of the river	No conditions	False	None
4	The farmer brings the cabbage on the bank B of the river	No conditions	False	None
5	The farmer brings the goat on the bank B of the river	No conditions	False	None
6	The farmer goes to the bank A of the river	It happened	True	None
7	The farmer brings the cabbage on the bank B of the river	It happened	True	None
8	The farmer brings the goat on the bank B of the river	No conditions	False	None
9	The farmer brings the wolf on the bank B of the river	No conditions	False	None
10	The farmer goes to the bank B of the river	No conditions	False	None
11	The farmer brings the cabbage on the bank B of the river	No conditions	False	None
12	The farmer goes to the bank A of the river	No conditions	False	None
13	The farmer brings the cabbage on the bank A of the river	The event could have happened, but caused a cycle	The event could have changed the state	None

Step	Event	Outcome	State changed?	Effects
14	The farmer brings the goat on the bank A of the river	It happened	True	None
15	The farmer brings the cabbage on the bank B of the river	No conditions	False	None
16	The farmer brings the goat on the bank B of the river	The event could have happened, but caused a cycle	The event could have changed the state	None
17	The farmer brings the wolf on the bank B of the river	It happened	True	None
18	The farmer goes to the bank B of the river	No conditions	False	None
19	The farmer brings the cabbage on the bank B of the river	No conditions	False	None
20	The farmer brings the goat on the bank B of the river	No conditions	False	None
21	The farmer brings the wolf on the bank B of the river	No conditions	False	None
22	The farmer goes to the bank A of the river	It happened	True	None
23	The farmer brings the cabbage on the bank B of the river	No conditions	False	None
24	The farmer brings the goat on the bank B of the river	It happened	True	The farmer, the wolf, the goat and the cabbage are on the bank B of the river

The brutal force attack is recognizable by all attempts to check a cause-effect relationship, where the **cause** event does not have the necessary conditions to occur, as in the case of the third attempt (cycle 2). Indeed, at the second attempt (cycle 1) Joshua checked the event "The farmer brings the wolf on the bank B of the river", the latter happened and changed the state (the Farmer and the wolf are on the bank B now), even if the effect (the solution) dit not occur. Therefore the third attempt can be seen as foolish, because Joshua checked the event "The farmer brings the wolf on the bank B of the river", that is impossible because the Farmer is already on the river B.

Note that the same event can occur multiple times along the sequence of attempts made by Joshua.

The reason lies in the Joshua's way of thinking. An attempt is a single chain of events, and if one chain failed to occur then Joshua forget it. Nevertheless, if at least one event of a chain occurred and the state is changed, then Joshua rehabilitates the previous checked chains.

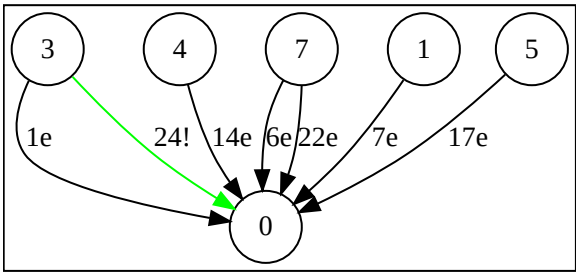
Steps 13 and 16 highlight how Joshua reasons too. Even if those events could have happened, Joshua discarded them because they would have changed the current state into a configuration already reached in the past. Joshua considered the happening of those events a potential loop.

If you removed all failed attempts from the table, what remains are the steps toward the solution.

Table 3. "The farmer, the wolf, the goat and the cabbage"

Step	Cause	effect
1	The farmer brings the goat on the bank B of the river	The state changed
6	The farmer goes to the bank A of the river	The state changed
7	The farmer brings the cabbage on the bank B of the river	The state changed
14	The farmer brings the goat on the bank A of the river	The state changed
17	The farmer brings the wolf on the bank B of the river	The state changed
22	The farmer goes to the bank A of the river	The state changed
24	The farmer brings the goat on the bank B of the river	The farmer, the wolf, the goat and the cabbage are on the bank B of the river

The nature of the approach visibly arise looking at the below diagram, that shows the occurrence of the cause-effect relationships.



To simplify the diagram, the events have been numbered:

- 0 - "The farmer, the wolf, the goat and the cabbage are on the bank B of the river" (**solution**)
- 1 - "The farmer brings the cabbage on the bank B of the river"
- 3 - "The farmer brings the goat on the bank B of the river"
- 4 - "The farmer brings the goat on the bank A of the river"
- 5 - "The farmer brings the wolf on the bank B of the river"
- 7 - "The farmer goes to the bank A of the river"

The diagram shows 5 different chains of concatenated event, each one containing only one cause-effect relationship. To derive the sequence of attempts, you have to watch the number of cycle associated to each arrow:

- cycle 1 - "The farmer brings the goat on the bank B of the river" occurred, but not the solution
- cycle 6 - "The farmer goes to the bank A of the river" occurred, but not the solution
- cycle 7 - "The farmer brings the cabbage on the bank B of the river" occurred, but not the solution
- cycle 14 - "The farmer brings the goat on the bank A of the river" occurred, but not the solution
- cycle 17 - "The farmer brings the wolf on the bank B of the river" occurred, but not the solution
- cycle 22 - "The farmer goes to the bank A of the river" occurred, but not the solution
- cycle 24 - "The farmer brings the goat on the bank A of the river" occurred, as well as the solution!

Even if the occurrence of the chains dit not cause the solution, they changes the state util eventually the last chain cause the solution.

[2] Cfr. https://en.wikipedia.org/wiki/YAML

3. Other problems

This chapter shows other problems which highlight some interesting aspects about how Joshua works.

3.1. The heritage

> A person gets an heritage of 1024€, and it starts spending half of the amount each day. How many days does he take to stay with less than 1 euro?
>
> — anonymous

A simple mathematical problem that can be easily described by three events:

- "He has less than 1 euro" is the solution event;
- "One day of spending"
- "The inheritance is halved"

```
knowlegde:
  - id: "One day of spending"
    conditions:
      - Inheritance > 1.0
    effects:
      - weight: 1.0
        event: "The inheritance is halved"
      - weight: 0.5
        event: "He has less than 1 euro"

  - id: "The inheritance is halved"
    effects:
      - weight: 0.5
        event: "He has less than 1 euro"
    assignments:
      - Inheritance = Inheritance / 2

  - id: "He has less than 1 euro"
    conditions:
      - Inheritance < 1.0
```

In this problem there is an always true cause-effect, indeed if you have "one day of spending", surely your inheritance will be halved.

Finally, the state only contains only the variable "Inheritance", with an initiali value of _1024: at the beginning of the problem.

```
joshua.exe solve -k k_euro.yml -p p_euro.yml
Max cycles:  100
State contains:  1  variables
        Outcome|   true|
         Cycles|     10|
   Queue's size|     12|
```

3.2. The water containers

> There are three containers A, B, C. A can hold a maximum of 3 litres of water, and it is empty. B can hold a maximum of 5 litres of water, and it is empty. C can hold a maximum of 8 litres of water, and it is full. There are no marks on the containers. You need to have at least one of B or C with 4 litres of water, how can you do that?
>
> — anonymous

This problem is really similar to that of the *Farmer*. Describing the events essentially means listing all possibile actions, as well as every action can potentially (not surely) causes the solution's effect.

```
knowlegde:
  - id: "Empty C in A"
    conditions:
      - C > 0
    effects:
      - weight: 0.5
        event: "B or C contains 4"
    assignments:
      - DeltaA = MaxA - A
      - SubC = min(C, DeltaA)
      - C = C - SubC
      - A = A + SubC

  - id: "Empty C in B"
    conditions:
      - C > 0
    effects:
      - weight: 0.5
        event: "B or C contains 4"
    assignments:
      - DeltaB = MaxB - B
      - SubC = min(C, DeltaB)
      - C = C - SubC
      - B = B + SubC

  - id: "Empty A in B"
    conditions:
```

```
      - A > 0
    effects:
      - weight: 0.5
        event: "B or C contains 4"
    assignments:
      - DeltaB = MaxB - B
      - SubA = min(A, DeltaB)
      - A = A - SubA
      - B = B + SubA

  - id: "Empty A in C"
    conditions:
      - A > 0
    effects:
      - weight: 0.5
        event: "B or C contains 4"
    assignments:
      - DeltaC = MaxC - C
      - SubA = min(A, DeltaC)
      - A = A - SubA
      - C = C + SubA

  - id: "Empty B in A"
    conditions:
      - B > 0
    effects:
      - weight: 0.5
        event: "B or C contains 4"
    assignments:
      - DeltaA = MaxA - A
      - SubB = min(B, DeltaA)
      - B = B - SubB
      - A = A + SubB

  - id: "Empty B in C"
    conditions:
      - B > 0
    effects:
      - weight: 0.5
        event: "B or C contains 4"
    assignments:
      - DeltaC = MaxC - C
      - SubB = min(B, DeltaC)
      - B = B - SubB
      - C = C + SubC

  - id: "B or C contains 4"
    conditions:
      - B == 4 || C == 4
```

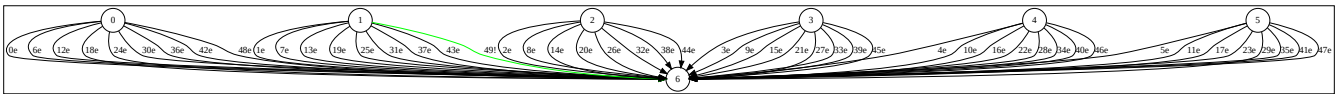
Unlike the *Farmer's* problem, here the assignments are not trivial but tricky, because their definition may look like anticipating reasoning to Joshua. Such consideration is also visible into the problem definition file.

```
variables:
  - name: A
    value: 0
    defined: true
  - name: B
    value: 0
    defined: true
  - name: C
    value: 8
    defined: true
  - name: MaxA
    value: 3
    defined: true
  - name: MaxB
    value: 5
    defined: true
  - name: MaxC
    value: 8
    defined: true
  - name: DeltaA
    value: 0
    defined: true
  - name: DeltaB
    value: 0
    defined: true
  - name: DeltaC
    value: 0
    defined: true
  - name: SubA
    value: 0
    defined: true
  - name: SubB
    value: 0
    defined: true
  - name: SubC
    value: 0
    defined: true
success: "B or C contains 4"
```

As you can see, beyond the obvious variables (A, B, C, D) about the amount of water into the containers, there are other variables useful only for temporary calculations.

Likewise the *Farmer's* problem, there is trial-and-error approach.

```
joshua.exe solve -k k_acqua.yml -p p_acqua.yml
Max cycles:  100
State contains:  12  variables
        Outcome|   true|
         Cycles|     49|
   Queue's size|     55
```



3.3. Flights and pilots

> Carlo, Bruno and Alberto work in the same team. One is the pilot, one is the copilot, and one is the engineer. The copilot does not have sisters and brothers, and he earns the least of all. Carlo is the husband of the Bruno's sister, and he earns more than the pilot. Can you say the job of everyone?
>
> — anonymous

A logic problem is probably less intuitive to describe, but it is really interesting, since it highlights Joshua's ability to exploit the concatenation of cause-effect relationships.

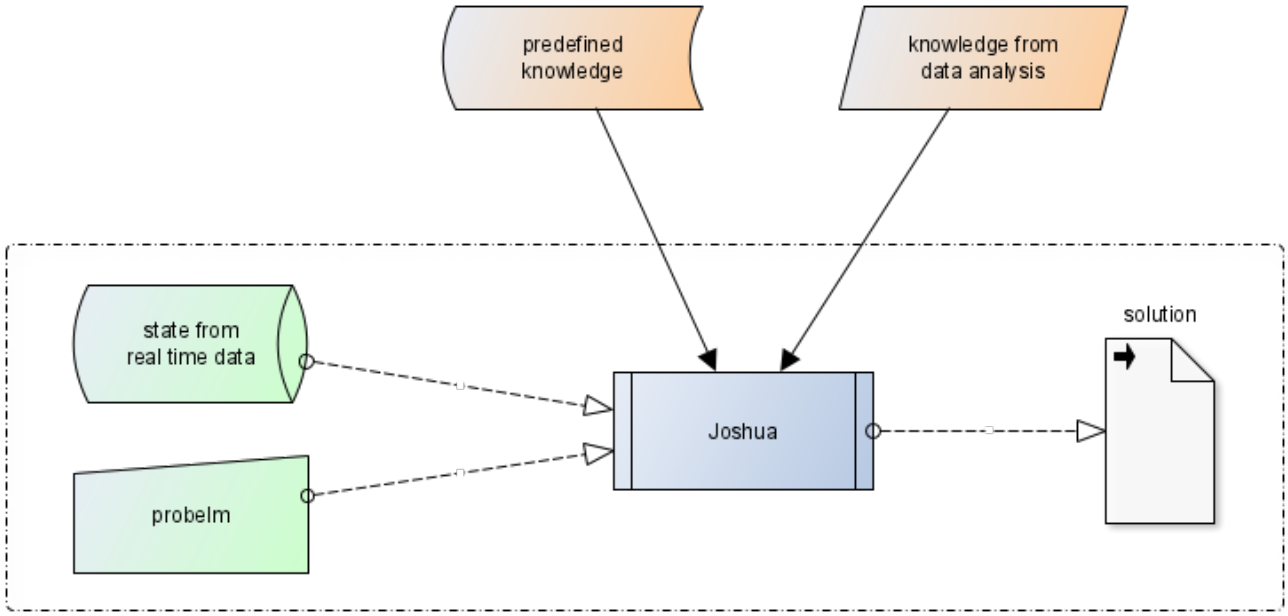
TBD

4. Dynamic knowledge and real time problems

In the previous chapters, a predefined knowledge was build to find out the one solution of a problem. Nevertheless, some problems may have different solutions and such solutions may depend on time.

For example, assuming to use Joshua in the financial trading, problems could be defined by events like "The stock is undervalued" and "Sell signal". In those cases the knowledge should not be static, at least the problem should have an initial state that depends on time.

Below diagram shows a possible approach.



The knowledge includes a set of predefined cause-effect relationships and relationships coming from a real time analysis of data. The initial state is the result of a process of gathering and analysis of real time data. Finally the problem is decided on the spot.

Just as an example, below a hypothetic set of predefined financial cause-effect relationships.

```
  knowlegde:
    - id: "P/B less or equal to 1"
      # This ratio compares the current price of stocks with the book value per share
of the stock.
      # Look for the book value per share on the company□s balance sheet or on a stock
website. Ratios under 1 are undervalued.
      conditions:
        - PB <= 1.0
      effects:
        - weight: 0.5
          event: "The stock is undervalued"
      assignments:
```

```
        - Undervalued = Undervalued + 1

    - id: "debt-to-asset ratio of 1.10 or less"
      # This means that they have more assets than they have debt. This is a sign of a
strong company and a good stock.
      conditions:
        - DA <= 1.10
      effects:
        - weight: 0.5
          event: "The stock is undervalued"
      assignments:
        - Undervalued = Undervalued + 1

    - id: "P/E less than sector reference"
      # This means that they have more assets than they have debt. This is a sign of a
strong company and a good stock.
      conditions:
        - PE < PE_ref
      effects:
        - weight: 0.5
          event: "The stock is undervalued"
      assignments:
        - Undervalued = Undervalued + 1

    - id: "PEG less than sector reference"
      # A stock's PEG ratio is the stock's P/E ratio divided by the growth rate of its
earnings.
      conditions:
        - PEG < PEG_ref
      effects:
        - weight: 0.5
          event: "The stock is undervalued"
      assignments:
        - Undervalued = Undervalued + 1

    - id: "Price-to-book less than sector reference"
      # The price to book is calculated by share price divided by book value per
share.
      conditions:
        - PBook < PBook_ref
      effects:
        - weight: 0.5
          event: "The stock is undervalued"
      assignments:
        - Undervalued = Undervalued + 1

    - id: "RSI greater then 0.7"
      conditions:
        - RSI > 0.7
      effects:
        - weight: 0.5
```

```
          event: "The stock is Overbought"
      assignments:
        - Overbought = Overbought + 1

    - id: "RSI is crossing over 0.5"
      conditions:
        - RSI_crossing_up == True
      effects:
        - weight: 0.5
          event: "Sell signal"
      assignments:
        - Sell_signal = Sell_signal + 1

    - id: "RSI lesser then 0.3"
      conditions:
        - RSI < 0.3
      effects:
        - weight: 0.5
          event: "The stock is Oversold"
      assignments:
        - Oversold = Oversold + 1

    - id: "RSI is crossing under 0.5"
      conditions:
        - RSI_crossing_down == True
      effects:
        - weight: 0.5
          event: "Buy signal"
      assignments:
        - Sell_signal = Sell_signal + 1

    - id: "The stock is Overbought"
      # Overbought is a term used when a security is believed to be trading at a level
above its intrinsic or fair value.
      effects:
        - weight: 0.5
          event: "Sell signal"

    - id: "The stock is Oversold"
      # When a security is thought to be trading below its intrinsic value
      effects:
        - weight: 0.5
          event: "Buy signal"

    - id: "The stock is Oversold"
      # When a security is thought to be trading below its intrinsic value

    - id: "The stock is undervalued"

    - id: "The stock is overvalued"
```

```
    - id: "Sell signal"

    - id: "Buy signal"

    - id: "Hold signal"
```

5. User guide

Joshua is available for the following architectures:

- Windows amd64
- Linux amd64
- ARM amd64
- Mac OS/X

The software is available for download at the link:

?????

5.1. How to write a knowledge YAML file

5.2. How to write a problem YAML file

5.3. CLI usage

```
joshua.exe solve -k k_contadino.yml -p p_contadino.yml -m 50 --queuelog
$ ./joshua.exe solve -k ../resources/k_contadino.yml -p ../resources/p_contadino.yml
-m 50 --queuelog
```

Appendix

https://ncona.com/2020/06/create-diagrams-with-code-using-graphviz/
http://www.webgraphviz.com/

- https://coderbook.com/@marcus/how-to-render-markdown-syntax-as-html-using-python/