CF Use - Solve '3+?=8' without Lego

1. CF inhibitor detects it's going nowhere
   ↳ Going Nowhere Event → WM
2. ↳ Processor detects this and activates more thorough solution strategy.
3. Searches LTM for strategies that may help.
   ↳
   1. Try to figure out what the unknown token may represent.
      ↳ compare to other known things / patterns
      ↳ looks like: 3+5=?
   2. Once have an idea, try it and see where it goes.
      ⇒ If, after a while it's going nowhere, go back to (1), and try something else.

Don't use too many keys on events. Use presence of other events together in WM.

Need the representation of keys in LTM?

Assume a priority of possible explanations for '?'.

Use entries in WM to remember what attempts have been made to avoid repeating them.

2/

Solve — '3+? =9' without help.

Going Nowhere ~~Infer From Similar~~ → ("3+?=9" ⇒ "[number] [op] [token] [=] [number]

→ Attempt To Parse

Just looks at each token in turn and takes strongest guess. eg: it knows what a number looks like

→ Infer From Similar:

"3+?=9" ⇒ No match in LTM

"[no.] [op] [token] [=] [number]"

⇒ Search LTM

⇒ Std. Expression.

⇒ Std. Expression With Result

Firstly:

⇒ [number] [op] [number] [=] [number]

⇒ Infer: '?' = [number] if it can find an exact match.

⇒ Otherwise it may only infer that "3+?=8" is an (unknown) expression.

- LTM: StdExpression
  [number] [op] [number]
  OR
  [op] [number]
  OR
  [op] ([number])

- LTM: Fact("number")
  - Referenceable or generic reference to the learned concept of a "number".

- Data Rules:
  Need a new 'primitive' type: ~~Guid~~ Ref.
  ← holds string 'guid' to another memory item or event.

Goal: should be able to use the same process to infer that "3 | 4" is some kind of operator.

Upon success, DayDreaming will help put sols into LTM as learnings.

Solve '3+3 = 8' without help

- So, now have:
  "3+3=8" is "[no.] [op.] [no.] [=] [no.]
  =3 =+ =unknown =8
  Fact
  Assumes in range 1..100.
- Pull ... (see below)
- Strategies:
  - Pick random number ← useful for guess the number games.
  - try each in sequence.
  - try to infer from "what y gives x?" learnings.

Need 'Range' in WM.

- Infer Number Range Processor
  ↳ Looks for [number] in WM and adds a RangeEvent to WM with a link to that [number]. — may add later.
- Pull [number] =? out of Expr into WM on its own — need to solve this, with link back to particular token in Expr.

For long term solutions, will need a Perseverance Processor of some sort that helps decide course of action at each point:
- ask for confirmation
- or assume inference is correct & carry on.

Range: long term solution would be to add an extra step of using a processor + LTM to infer range from various things. But for now just hard-code in a processor.

For DejaVu & GoingNowhere detection, look at:
- are we going in loops?
- is there benefit in keeping going? (ie: for i in 1..100 & i = 50)
- adjusted sense of limits (ie: 100 tries is max; previously decided need to try 500 times; been instructed to try 50 times).

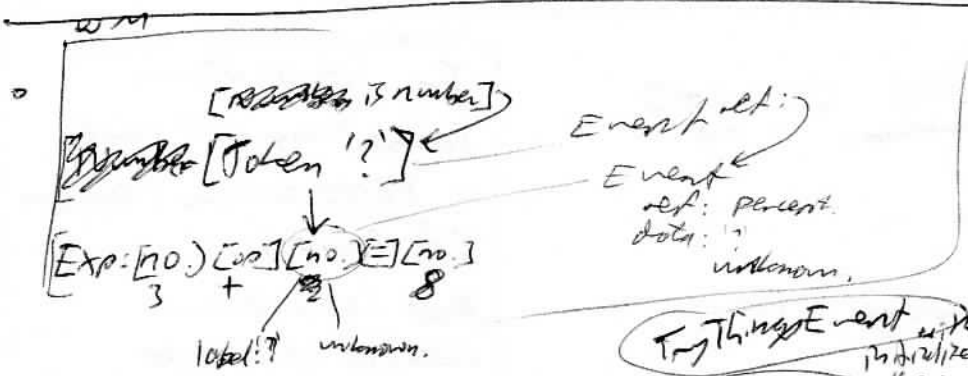
Dealing with trying multiple solutions:
- handled by WM + STM.
- Processors that infer solutions attenuated by A-A(?) if their prior solutions didn't pan out, as recorded in STM or WM

Phenomena: when trying for another solution, I immediately ignore not try retry my immediate previous attempt but I may have to think a bit more to check I didn't try something in the past.

4/

Solve '3+?=8' w/o help.

WM



TryThingsEvent with initialized "Tried" data.

⇒ Going Nowhere → TryThings suitable to numbers.

Quick Impl.

Processors that know how to try:

- pick random
- look sequence range + current position.
- pick next in sequence

AA just picks winner each time.

Long Impl.

LTM lookup([number], strategies to solve)

→ WM: list of possible strategies

⇓

Processor picks most likely strategy that hasn't already been tried.

↓

Check WM for "Already Tried" list, ← should start as present, but with empty list.

or re-load from STM

- strategies themselves are pre-learned by processors & LTM entry just identifies them.
... or could be algorithmic memory item.

may need a Concept type.

eg: [no.] [op.] [no.] [=] [no.]

Concepts

eg: [Concept
.Data = 3
.label = '?'
.unknown = true
.ref

↳ [concept
.number
.
↕
[concept: spoken word
"number"

How about:

Percept — class

Event — instance.

Wed 9am. 11am