Factbase NetLogo Extension Version 1.1

What is it?

Factbase is an extension for NetLogo (version 6.1) that introduces a new data type: a structured set of data called a "fact base". A fact base can be thought of as a table of named columns ("fields"), where each row comprises an entry ("fact"). Facts in this extension are represented as lists of values, with one value per field and all values in the correct order as defined by the structure of the corresponding fact base.

The name fact base – instead of data base – was chosen in order to avoid any misconceptions since this extension does not involve the use of data base management systems. We are aware that this choice of name might cause confusion in another direction, namely with logical programming. The understanding of facts and fact bases used in this extension (facts consist of field values; all facts in a fact base have the same fields) differs slightly from that used in logical programming (facts consist of a name and field values; facts in a fact base can have different fields). Nevertheless, we decided that the commonalities are strong enough to warrant the name fact base.

Primitives

factbase:create <list-of-field-names>

Reports a new empty fact base with the given structure. Example:

```
set calendar factbase:create ["when" "what" "where" "who"]
show calendar
> {{factbase: [["when" "what" "where" "who"]]}}
```

factbase:assert <fbase> <fact>

Asserts the given fact to the specified fact base if (a) it has the correct structure and (b) it is not already contained within the fact base. Incorrect structure will cause an error, while trying to assert a duplicate fact will just be ignored. Example:

```
factbase:assert calendar (list (ticks + 2) "meeting friends" the-pub (turtle 23))
factbase:assert calendar (list (ticks + 4) "dinner" my-home myself)
```

factbase:assert-all <fbase> <list-of-facts>

Asserts all of the given facts to the specified fact base. Each fact is checked if (a) it has the correct structure and (b) it is not already contained within the fact base. Example:

```
factbase:assert-all calendar [[12 "work-out" the-gym myself][14 "lunch" the-pub (turtle 4)][15 "check-up" the-dentist myself]]
```

factbase:size <fbase>

Reports the number of facts in the specified fact base. Example:

```
show factbase:size calendar
```

factbase:member? <fbase> <fact>

Reports *true*, if the given fact is contained in the specified fact base; otherwise, reports *false*. Example:

```
show factbase:member? calendar [19 "dinner" my-home myself]
```

factbase:retrieve <fbase> <condition-task> <list-of-referred-fields>

Reports a list of facts from the specified fact base that satisfy the given condition. The condition is given as a reporter task with one argument for each field to be used in the query. A list of field names specifies which argument refers to which field. Note that you can name the variables in the reporter task freely, e.g. "?1" or "t". Examples:

```
let current-events factbase:retrieve calendar [t -> t = ticks]
["when"]
```

```
let past-meetings-with-friends factbase:retrieve calendar [[a t]
-> a = "meeting friends" and t < ticks] ["what" "when"]
```

```
let meet-Fred-in-future factbase:retrieve calendar [[?1 ?2] -> ?1
= (turtle 4) and ?2 > ticks] ["who" "when"]
```

factbase:retrieve-to <fbase> <condition-task> <list-of-referred-fields> <list-of-output-fields>

Same as retrieve, but instead of whole facts, reports only the specified fields (*list-of-output-fields*) for each fact that satisfies the given condition. Example:

```
let current-events factbase:retrieve-to calendar [t -> t = ticks]
["when"] ["what" "when"]
```

factbase:get <fbase> <fact-id>

Reports the fact with the given fact ID from the specified fact base. Fact IDs run from 0 to *size*-1. Facts in NetLogo are represented as lists with one element per field defined for the fact base. Example:

```
let fact factbase:get calendar 2 ;; retrieve third entry from
calendar
```

factbase:exists? <fbase> <condition-task> <list-of-referred-fields>

Reports *true* if there is at least one fact in the specified fact base that satisfies the given condition; otherwise, reports *false*. The structure of the condition follows the same rules as for fact base retrieval. Example:

```
factbase:exists? calendar [[?1 ?2] -> ?1 = the-dentist and ?2 =
myself] ["where" "who"]
```

factbase:retract <fbase> <fact>

Retracts the given fact from the specified fact base if (a) it has the correct structure and (b) is contained within the fact base. Example:

```
factbase:retract calendar (list 19 "dinner" my-home myself)
```

factbase:retract-all <fbase> <condition-task> <list-of-referred-fields>

Retracts all facts from the specified fact base that satisfy the given condition. The structure of the condition follows the same rules as for fact base retrieval. Example:

```
factbase:retract-all calendar [t -> t < ticks] ["when"]
```

factbase:to-list <fbase>

Reports the whole fact base as a list of lists (facts). The first element contains the field names. Example:

```
show factbase:to-list calendar
> [["when" "what" "where" "who"] [2 "meeting friends" (patch 2 2)
(turtle 23)] [4 "dinner" (patch 3 4) (turtle 0)] …]
```

factbase:from-list <list-of-fields-and-facts>

Creates a new fact base with the specified structure, asserts the given facts and then reports this fact base. The *list-of-fields-and-facts* has to be a list of lists, with the first element containing the field names and the rest of the list containing the initial facts, exactly like the result of the *to-list* primitive. This means it is possible to copy a fact base by using the output of *to-list* as an input to *from-list*. Example:

```
set calendar factbase:from-list (factbase:to-list [calendar] of
turtle 1)
```

factbase:one-of <fbase> <condition-task> <list-of-referred-fields>

Reports a random fact from the specified fact base satisfying the given condition. If there is no such fact in the specified fact base, generates an error. The structure of the condition follows the same rules as for fact base retrieval. Example:

```
let a-future-event factbase:one-of calendar [t -> t > ticks]
[“when”]
```

factbase:n-of <fbase> <condition-task> <list-of-referred-fields> <n>

Reports *n* random facts from the specified fact base satisfying the given condition. If there are fewer than *n* such facts in the specified fact base, generates an error. The structure of the condition follows the same rules as for fact base retrieval. Example:

```
let future-events factbase:n-of calendar [t -> t > ticks][“when”]
3
```

factbase:r-assert <fbase> <fact>

This is a variant of `factbase:assert` that reports the fact ID given to the newly asserted fact. In NetLogo terms, `assert` is a command while `r-assert` is a reporter. Examples:

```
show factbase:r-assert calendar (list (ticks + 2) "meeting
friends" the-pub (turtle 23))
> 8
let id factbase:r-assert calendar (list (ticks + 4) "dinner" my-
home myself)
```

Acknowledgements

Extension developed under the DiDIY Project funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644344. The views expressed in this article do not necessarily reflect the views of the EC.

Persistent Identifier: CPM-16-227