Types in the Power Query M formula language

The Power Query M Formula Language is a useful and expressive data mashup language. But it does have some limitations. For example, there is no strong enforcement of the type system. In some cases, a more rigorous validation is needed. Fortunately, M provides a built-in library with support for types to make stronger validation feasible.

Developers should have a thorough understanding of the type system in-order to do this with any generality. And, while the Power Query M language specification explains the type system well, it does leave a few surprises. For example, validation of function instances requires a way to compare types for compatibility.

By exploring the M type system more carefully, many of these issues can be clarified, and developers will be empowered to craft the solutions they need.

Knowledge of predicate calculus and naïve set theory should be adequate to understand the notation used.

PRELIMINARIES

(1) $B := \{ \textit{true; false} \}$

B is the typical set of Boolean values

(2) $N := \{$ valid M identifiers $\}$

N is the set of all valid names in M. This is defined elsewhere.

(3) $P := \langle B,T \rangle$

P is the set of function parameters. Each one is possibly optional, and has a type. Parameter names are irrelevant.

(4) $P^n := \bigcup_{0 \le i \le n} \langle i, P \rangle$

P^n *is the set of all ordered sequences of n function parameters*

(5) $P^* := \bigcup_{0 \le i \le \infty} P^i$

P is the set of all possible sequences of function parameters, from length 0 on up.*

(6) $\mathrm{F} := \langle \mathrm{B}, \mathrm{N}, \mathrm{T} \rangle$

F is the set of all record fields. Each field is possibly optional, has a name, and a type.

(7) $\mathrm{F}^n := \prod_{0 \le i \le n} \mathrm{F}$

F^n *is the set of all sets of n record fields*

(8) $\mathrm{F}^* := \left(\bigcup_{0 \le i \le \infty} \mathrm{F}^i \right) \setminus \{F \mid \langle b_1, n_1, t_1 \rangle, \langle b_2, n_2, t_2 \rangle \in F \ \wedge \ n_1 = n_2\}$

F^* *is the set of all sets (of any length) of record fields,* ***except for*** *the sets where more than one field has the same name.*

(9) $\mathrm{C} := \langle \mathrm{N}, \mathrm{T} \rangle$

C is the set of column types, for tables. Each column has a name and a type.

(10) $\mathrm{C}^n \subset \bigcup_{0 \le i \le n} \langle i, \mathrm{C} \rangle$

C^n *is the set of all ordered sequences of n column types.*

(11) $\mathrm{C}^* := \left(\bigcup_{0 \le i \le \infty} \mathrm{C}^i \right) \setminus \{C^m \mid \langle a, \langle n_1, t_1 \rangle \rangle, \langle b, \langle n_2, t_2 \rangle \rangle \in C^m \wedge n_1 = n_2\}$

C is the set of all combinations (of any length) of column types,* ***except for*** *those where more than one column has the same name*

M TYPES

(12) $\mathrm{T_F} := \langle \mathrm{P}, \mathrm{P}^* \rangle$

A Function Type consists of a return type, and an ordered list of zero-or-more function parameters

(13) $\mathrm{T_L} := \ [\![\mathrm{T}]\!]$

A List type is indicated by a given type (called the "item type") wrapped in curly braces. Since curly braces are used in the metalanguage, ⟦ ⟧ *brackets are used in this document.*

(14) $T_R := \langle B, F^* \rangle$

A Record Type has a flag indicating whether it's "open", and zero-or-more unordered record fields.

(15) $T_R^o := \langle true, F^* \rangle$

(16) $T_R^\bullet := \langle false, F^* \rangle$

T_R^o and $T_R^\bullet$ are notational shortcuts for open and closed record types, respectively.

(17) $T_T := C^*$

A Table Type is an ordered sequence of zero-or-more column types, where there are no name collisions.

(18) T_P := { any; none; null; logical; number; time; date; datetime; datetimezone; duration; text; binary; type; list; record; table; function; anynonnull }

A Primitive Type is one from this list of M keywords.

(19) $T_N := \{t_n, u \in T \mid t_n = u + \text{null}\} = \text{nullable } t$

Any type can additionally be marked as being nullable, by using the "nullable" keyword.

(20) $T := T_F \cup T_L \cup T_R \cup T_T \cup T_P \cup T_N$

The set of all M types is the union of these six sets of types:
Function Types, List Types, Record Types, Table Types, Primitive Types, and Nullable Types.

FUNCTIONS

One function needs to be defined: NonNullable : T ← T

This function takes a type, and returns a type that is equivalent except it does not conform with the null value.

IDENTITIES

Some identities are needed to define some special cases, and may also help elucidate the above.

(21) nullable any = any
(22) nullable anynonnull = any
(23) nullable null = null
(24) nullable none = null
(25) nullable nullable $t \in$ T = nullable t
(26) $NonNullable$(nullable $t \in$ T) = $NonNullable(t)$
(27) $NonNullable$(any) = anynonnull

TYPE COMPATIBILITY

As defined elsewhere, an M type is compatable with another M type if and only if all values that conform to the first type also conform to the second type.

Here is defined a compatability relation that does not depend on conforming values, and is based on the properties of the types themselves. It is anticiplated that this relation, as defined in this document, is completely equivalent to the original semantic definition.

The "is compatible with" relation $\leq$: B←T×T

In the below section, a lowercase t will always represent an M Type, an element of T.

A Φ will represent a subset of F^*, or of C^*.

(28) $t \leq$ t

This relation is reflexive

(29) $t_a \leq t_b \wedge t_b \leq t_c \rightarrow t_a \leq t_c$

This relation is transitive

(30) none $\leq t \leq$ any

M types form a lattice over this relation; none *is the bottom, and* any *is the top.*

(31) $t_a, t_b \in T_N \wedge t_a \leq t_b \rightarrow NonNullable(t_a) \leq NonNullable(t_b)$

If two types are compatible, then the NonNullable equivalents are also compatible

(32) $\text{null} \leq t \in T_N$

The primitive type null *is compatible with all nullable types*

(33) $t \notin T_N \leq \text{anynonnull}$

All nonnullable types are compatible with anynonnull

(34) $NonNullable(t) \leq t$

A NonNullible type is compatible with the nullable equivalent

(35) $t \in T_F \rightarrow t \leq \text{function}$

All function types are compatible with function

(36) $t \in T_L \rightarrow t \leq \text{list}$

All list types are compatible with list

(37) $t \in T_R \rightarrow t \leq \text{record}$

All record types are compatible with record

(38) $t \in T_T \rightarrow t \leq \text{table}$

All table types are compatible with table

(39) $t_a \leq t_b \leftrightarrow \ [\![t_a]\!] \ \leq \ [\![t_b]\!]$

A list type is compaible with another list type if the item types are compatible, and vice-versa

(40) $t_a \in T_F = \langle p_a, p^* \rangle, t_b \in T_F = \langle p_b, p^* \rangle \wedge p_a \leq p_b \rightarrow t_a \leq t_b$

A function type is compatible with another function type if the return types are compatible, and the parameter lists are identical

(41) $t_a \in T_R^o, t_b \in T_R^\bullet \rightarrow t_a \not\leq t_b$

An open record type is never compatible with a closed record type

(42) $t_a \in T_R^\bullet = \langle false, \Phi \rangle, t_b \in T_R^o = \langle true, \Phi \rangle \rightarrow t_a \leq t_b$

A closed record type is compatible with an otherwise identical open record type

(43) $t_a \in T_R^o = \langle true, (\Phi, \langle true, n, any \rangle) \rangle, t_b \in T_R^o = \langle true, \Phi \rangle \rightarrow t_a \leq t_b \wedge t_b \leq t_a$

An optional field with the type any *may be ignored when comparing two open record types*

(44) $t_a \in T_R = \langle b, (\Phi, \langle \beta, n, u_a \rangle) \rangle, t_b \in T_R = \langle b, (\Phi, \langle \beta, n, u_b \rangle) \rangle \wedge u_a \leq u_b \rightarrow t_a \leq t_b$

Two record types that differ only by one field are compatible if the name and optionality of the field are identical, and the types of said field are compatible.

(45) $t_a \in T_R = \langle b, (\Phi, \langle false, n, u \rangle) \rangle, t_b \in T_R = \langle b, (\Phi, \langle true, n, u \rangle) \rangle \rightarrow t_a \leq t_b$

A record type with a non-optional field is compatible with a record type identical but for that field being optional.

(46) $t_a \in T_R^o = \langle true, (\Phi, \langle b, n, u \rangle) \rangle, t_b \in T_R^o = \langle true, \Phi \rangle \rightarrow t_a \leq t_b$

An open record type is compatible with another open record type with one fewer field

(47) $t_a \in T_T = (\Phi, \langle i, \langle n, u_a \rangle \rangle), t_b \in T_T = (\Phi, \langle i, \langle n, u_b \rangle \rangle) \wedge u_a \leq u_b \rightarrow t_a \leq t_b$

A table type is compatible with a second table type, which is identical but for one column having a differing type, when the types for that column are compatible.

REFERENCES

Microsoft Corporation (2015 August)
Microsoft Power Query for Excel Formula Language Specification [PDF]
Retrieved from https://msdn.microsoft.com/en-us/library/mt807488.aspx

Microsoft Corporation (n.d.)
Power Query M function reference [web page]
Retrieved from https://msdn.microsoft.com/en-us/library/mt779182.aspx