C The Q Requirements Specification Language

This report introduces a specification making use of certain requirements specification language features. These features are combined to form Q, a requirements specification language. This section describes the motivation for Q and defines the Q specification language.

C.1 Overview

To be practical, the syntax of the specification language should impose as few restrictions on specification authors as possible. It may be necessary to produce test frames for a specification which is an arbitrary relation between any number of stimuli, pre- and post-conditions, and responses. This approach is different from techniques based on specification styles such as Z [12] or VDM [7] which require specifications to be broken down into pieces to be modeled in terms of schemata or operations.

The objective of Q is to provide a means of formalizing requirements phrases while maintaining readability and conciseness as much as possible. The formal aspect of the language is required by the test frame generator. A formal specification that is also readable relieves the need for maintaining two specifications; one formal for input to the test frame generator and another readable by non-specialists. Thus, a key design issue is that Q must be readable by non-specialists.

The specification language provides a concise syntax for denoting the logical relationships and alternatives within the requirements while also providing a natural language style. For example, the requirement fragment,

Either the leading aircraft or the trailing aircraft is supersonic

is specified as

```
{any of {the leading aircraft, the trailing aircraft}} is supersonic.
```

The braces impose a parseable structure on the requirements. The semantics of the language constructs, such as `''any of,''` allows the test frame generator to calculate the logically equivalent expression, which in this case is:

```
{{the leading aircraft} is supersonic} or {{the trailing aircraft}
is supersonic}.
```

Q is implemented as an extension of the S specification language [9] and is used to formalize natural language stimulus-response style specifications for the purpose of requirements-based testing. Q can be used to define predicates within a requirements specification but relies on S syntax for defining constants, type, and functions. Q statements are contained within the keywords BEGIN_Q and END_Q.

The light-weight simplicity of the Q language helps to preserve the readability and conciseness of the specification. The mathematical semantics of Q ensure that each statement has an unambiguous meaning. With these qualities, Q provides the mathematical link between a requirements specification and the test frame generation tool introduced in the previous chapter.

Three features of Q substantially contribute to preserving readability and conciseness. The first is the use of braces, {}, which delimit phrases and parameters within the specification. Injecting these braces into the specification effectively transforms the phrases of natural language into formal functions and arguments. This technique was first used by Joyce in his Test Case Element Language (TCEL) [8].

When formalizing the natural language phrase

> the leading aircraft is supersonic or the following aircraft is supersonic

for the purpose of system-level requirements-based testing, only the choices need to be made explicit. Thus, the appropriate formalization for testing is to choose `''or''` as the predicate and the two adjoining phrases are conditions. The resulting Q version of the above phrase is:

> `{the leading aircraft is supersonic} or {the following aircraft is supersonic}.`

In this Q expression, `'' * or * ''` is the function and `''the leading aircraft is supersonic''` and `''the following aircraft is supersonic''` are its arguments. The lambda calculus equivalent is

> `'' * or * '' ''the leading aircraft is supersonic'' ''the following aircraft is supersonic''`

where `'' * or * ''` has the type $\texttt{bool} \rightarrow \texttt{bool} \rightarrow \texttt{bool}$, as expected.

The `'' * ''` in the function name denotes positions in the text where arguments are placed. This type of notation is referred to as a flex-fix notation [1]. Flex-fix, the second Q feature, allows arguments to be distributed within a function name. This helps preserve readability. For example, the Q expression

> `''{aircraft A} and {aircraft B} are separated by at least {1000 feet}''`

corresponds to the following lambda calculus representation:

> `'' * and * are separated by at least * '' ''aircraft A'' ''aircraft B'' ''1000 feet.''`

The Q expression is more readable than, say, a Z or VDM-SL expression

$$ABS(Altitude(aircraft\_A) - Altitude(aircraft\_B)) \geqslant feet(1000).$$

The third feature of Q, due to the author, is the use of keywords which define multiple arguments for a function's parameter. These keywords are motivated by natural language phraseology such as "both aircraft are...," "either A or B is a... ." For example, the requirement

> either the leading aircraft or the following aircraft is supersonic

can be formalized in Q as

```
{any of {the leading aircraft, the following aircraft}} is supersonic.
```

A predicate containing an "any of" argument is equivalent to a disjunction of that predicate evaluated at each of the values in the "any of" set. In this case, the equivalent expression is

```
{{the leading aircraft} is supersonic} or {{the following aircraft}
is supersonic}.
```

This example contains more formal detail than the first example in this section. Here, there are formal references to two aircraft. In the first example, there were only two conditions. The fact that these conditions were based on two aircraft was not made explicit in the first example. This latest example is referred to as a deeper specification because it contains more formal detail. Test engineers decide how deep a specification should be by determining the condition dependencies they wish to reveal to the test frame generator.

Another parameter mechanism is the "distinct choices" keyword. This keyword is used in encoding phrases such as:

> all of the following are true:
>
> 1. aircraft A is dumping fuel,
> 2. aircraft B is using standard altimeter setting,
> 3. if one aircraft is supersonic and the other is not then ...

In this example, "one aircraft" and "the other" refer to either "aircraft A" or "aircraft B," interchangeably. i.e., They represent distinct choices of the two aircraft. The Q version is:

```
{all of {

  1. {aircraft A} is dumping fuel,

  2. {aircraft B} is using standard altimeter setting,

  3. if {{one aircraft, the other} are any distinct choices of {aircraft
     A, aircraft B} in
     {{{one aircraft} is supersonic} and {it is not the case that
     {{the other} is supersonic}}}} then...
```

The loss of conciseness in this example is necessary in order to formally define the references "one aircraft" and "the other." However, this construction is still more concise and more readable than the full expansion of the distinct choice which is:

```
{{{aircraft A} is supersonic} and
        {it is not the case that {{aircraft B} is supersonic}}}
or
{{{aircraft B} is supersonic} and
        {it is not the case that {{aircraft A} is supersonic}}}
```

This point is even more apparent when considering the case where a third aircraft is involved.

```
{one aircraft, another} are any distinct choices of {aircraft A,
aircraft B, aircraft C} in
{{{one aircraft} is supersonic} and {it is not the case that {{another}
is supersonic}}}
```

is equivalent to:

```
{{{aircraft A} is supersonic} and {it is not the case that {{aircraft
B} is supersonic}}} or
{{{{aircraft A} is supersonic} and {it is not the case that {{aircraft
C} is supersonic}}} or
{{{{aircraft B} is supersonic} and {it is not the case that {{aircraft
A} is supersonic}}} or
{{{{aircraft B} is supersonic} and {it is not the case that {{aircraft
C} is supersonic}}} or
{{{{aircraft C} is supersonic} and {it is not the case that {{aircraft
A} is supersonic}}} or
{{{{aircraft C} is supersonic} and {it is not the case that {{aircraft
B} is supersonic}}}}}}}
```

This example also demonstrates that the use of the distinct choices keyword helps preserve the understandability of the specification. Comprehending the Q expression above requires more effort than text which makes use of the `''distinct choices''` keyword.

The formal semantics of `''any of,''` its counterpart, `''each of,''` and other parameter mechanisms are more precisely defined in later sections.

C.2 Expressions

A Q expression is a string of at least one word and any number of arguments separated by white-space characters. Arguments are expressions contained within a comma delimited list surrounded by braces. In this thesis Q expressions are usually enclosed in braces to delimit them from the rest of the text. To be concise, an ambiguous grammar is used to express the syntax of Q. In the following grammar, e* represents zero or more e's concatenated, e.g., any of nil, e, ee, ..., where nil is the empty string. The expression e+ is equivalent to ee*. Parentheses are used to group expressions together in order to then apply * or +, e.g., a (";" b)* represents any of a, a;b, a;b;b, ...

expression	:=	word+ "." primitive_expression
	|	primitive_expresion
primitive_expression	:=	("{" expression ("," expression)* "}")+ primitive_expression
	|	primitive_expression ("{" expression ("," expression)* "}")+
	|	word+

The optional prefix for each expression allows specification authors to tag expressions for traceability purposes. These tags have no semantic value with respect to the logical meaning of the specification.

C.3 Predicate Definitions

A Q specification is a collection of predicate definitions. Predicates are defined using the `` * is true iff * '' statement.

definition	:=	"{" parm_expression "}" is true iff "{" expression "}" "."
parm_expression	:=	("{" word+ ("," word+)* "}")+ parm_expression
	|	parm_expression ("{" word+ ("," word+)* "}")+
	|	word+

C.4 Conjunctive and Disjunctive Lists

Requirements specifications often provide lists of conditions which represent logical conjunction, e.g., "all of the following," or disjunction, e.g., "at least one of the following." Such a list format is provided by the predicates ``all of'' and ``any of.'' The Q expression {all of {S}}, where S is a comma separated list of predicates, is semantically equivalent to $\bigwedge$S, where $\bigwedge(\{x\} \cup A) = x \wedge (\bigwedge A)$, and $\bigwedge\{\} = \top$. Similarly, {any of {S}} is semantically equivalent to $\bigvee$S, where $\bigvee(\{x\} \cup A) = x \vee (\bigwedge A)$, and $\bigvee\{\} = \bot$.

C.5 Argument Based Conjunctions and Disjunctions

The functions ``each of * '' and ``any of * '' are used to construct conjunctions and disjunctions, respectively, of a predicate over different arguments. These functions both have the type $(t)list \rightarrow t$. The semantics of these functions is defined in terms of predicates (Boolean expressions that do not contain logical connectives). The equivalent logic expression is determining by evaluating the Boolean expression $AE\_Ua\ P$ for ``any of'' or $AE\_Ue\ P$ for ``each of'' using axioms [4]. These two functions map the application of a predicate to a list of arguments into a disjunction or conjunction, respectively, of the predicate applied to each argument of the list, separately.

Although multiple uses of one of these keywords can be used within a predicate, mixtures of ``any of'' and ``each of'' within arguments to a single reference of a predicate are problematic. This is because it is unclear whether the expression containing argument keywords represents a conjunction of disjunctions or vice versa.

For example, the expression

```
{the {each of {apple, tomato}} is a {any of {vegetable, fruit}}}
```

may have been intended to mean either

```
{{{the {apple} is a {vegetable}} or {the {tomato} is a {vegetable}}}
and {{the {apple} is a {fruit}} or {the {tomato} is a {fruit}}}}
```

or, alternatively,

```
{{{the {apple} is a {vegetable}} and {the {tomato} is a {vegetable}}}
or {{the {apple} is a {fruit}} and {the {tomato} is a {fruit}}}}.
```

Clearly these two semantic evaluations are logically different.

Although the axioms for `''any of''` and `''each pf''` disambiguate such a construction, this rule would need to be learned and would not be obvious to a non-specialist from the text alone. Since this is counter to the objective of Q, mixtures of `''any of''` and `''each of''` are not allowed within arguments to the same predicate. The order of semantic evaluation in these situations can be made more clearly using expression aliasing.

C.6 Expression Aliasing

An expression alias is the same as the let statement found in functional programming languages such as ML [10]. The purpose of the alias is to assign a short name to a complex expression in order to make a portion of text more readable.

The Q expression `{{x} is {y} in {E}}` is semantically equivalent to `{E}` with `y` substituted for `x`. To encourage simpler specifications, the expression E must be a predicate. The predicate `{{x} is {y} in {E}}` is syntactic sugar for the lambda calculus expression $(\lambda\, \mathtt{x}.\mathtt{E})\mathtt{y}$. Similarly, the tuple form `{{x,y} are {a,b} in {E}}` is syntactic sugar for the lambda calculus expression $(\lambda\, \mathtt{x}, \mathtt{y}.\mathtt{E})(\mathtt{a}, \mathtt{b})$.

Using expression aliasing, the earlier `''any of''` / `''each of''` example can be disambiguated as

```
{{item} is {each of {apple, tomato}} in
{the {item} is a {any of {vegetable, fruit}}}}
```

which results in a conjunction of disjunctions.

C.7 Argument Permutation

The predicates `'' * are all distinct choices of * in * ''` and `'' * are any distinct choices of * in * ''` are used to construct conjunctions and disjunctions involving permutations of arguments. An example of the use of this keyword was given above.

`{{x} are all distinct choices of {A} in {E}}`

is syntactically equivalent to

`{{x} are {each of {P(A)}} in {E}},`

where `P(A)` is a list of all the permutations of tuples the same size as `x` uses elements of `A`. Similarly,

`{{x} are any distinct choices of {A} in {E}}`

is syntactically equivalent to

`{{x} are {any of {P(A)}} in {E}}.`

C.8 Quantification

Universal and existential quantification are provided by the syntactic forms `{for any {x} {E}}`, which is equivalent to $\forall$ `x.E`, and `{there exists {x} such that {E}}`, which is equivalent to $\exists$ `x.E`.