The Q Language

John A. Watlington

Feb. 2000 DRAFT

1 Introduction

This is a description of a programming language based on the upper (dataflow) level of the Q instruction set. While a "high level" language, it describes algorithms in a manner which can be directly mapped into Q program elements. It is based on the Multidimensional Lucid language, which is briefly presented in section 3 for comparison.

1.1 Syntax Notation

The syntax descriptions are given in Backus-Naur Format (BNF). A statement contains a term being defined, followed by a '**:=**', then the definition. Terms in a definition are grouped using parentheses ( ), and alternative options (one of which must be present) are separated by a vertical bar (|). Optional portions of the definition are indicated within square braces ([ ]). Constant portions of the definitions which must be present — keywords and separators — are shown in single quotes and a bold face font, *e.g.* '**)**' or '**if**'.

An *identifier* is a unique sequence of alphanumeric characters (including '_'), constrained to start with an alphabetic character. It is used to identify program elements. A *simple_identifier* is a single alphanumeric character, used for naming dimensions. A *constant* is a data value which is fixed and known when the algorithm is compiled.

2 Syntax

The basic language element is the *term*. It may be either an atom, or a *where clause* with a single term as its subject. A term N-tuple is a list of N terms. If N is greater than one, the curly braces ({ }) are required around the entire list.

The Q language supports recursively structured blocks of definitions, using *where clauses*. Each where clause may have an internally scoped set of dimensions, graphs and streams declared, which are used by one or more local definitions. A *graph* is typically a named where clause. The *size* of a where clause is defined as the number of terms in the clause subject.

A partial BNF description of the Q language is shown in Fig. 1. The remaining portions — the definitions of *stream_decls* and *func_decls* — are shown in Figs. 2 and 3 respectively.

$$
\begin{array}{rcll}
\mathit{program} & := & \mathit{where\_body} & (1) \\
\mathit{term} & := & \mathit{where\_clause\_of\_size\_1} \ \mid \ [\ \text{‘(’}\ ]\ \mathit{atom}\ [\ \text{‘)’}\ ] & (2) \\
\mathit{atom} & := & \mathit{constant} \ \mid \ \mathit{stream} \ \mid \ \mathit{operation} \ \mid \ \mathit{func\_call} & (3) \\
\mathit{term\_tuple} & := & \mathit{term}\ [\ \text{‘,’}\ \mathit{term\_tuple}\ ] & (4) \\
\mathit{stream} & := & \mathit{identifier} & (5) \\
\mathit{operation} & := & (\ \mathit{unary\_op}\ [\ \text{‘.’}\ \mathit{dims\_list}\ ]\ \mathit{term}\ ) \ \mid & \\
 & & (\ \mathit{term}\ \mathit{binary\_op}\ [\ \text{‘.’}\ \mathit{dims\_list}\ ]\ \mathit{term}\ ) \ \mid & \\
 & & (\ \text{‘}\textbf{if}\text{’}\ \mathit{cond\_expr}\ \text{‘}\textbf{then}\text{’}\ \mathit{term}\ [\ \text{‘}\textbf{else}\text{’}\ \mathit{term}\ ]\ \text{‘}\textbf{fi}\text{’}\ ) \ \mid & \\
 & & (\ \text{‘}\textbf{mcond}\text{(’}\ \mathit{cond\_list}\ \text{‘)’}\ ) & (6) \\
\mathit{dims\_list} & := & \mathit{dimension}\ [\ \text{‘,’}\ \mathit{dims\_list}\ ] & (7) \\
\mathit{dimension} & := & \mathit{simple\_identifier} & (8) \\
\mathit{unary\_op} & := & \text{‘}-\text{’} \ \mid \ \text{‘}\textbf{not}\text{’} \ \mid \ \text{‘}\textbf{first}\text{’} \ \mid \ \text{‘}\textbf{next}\text{’} \ \mid \ \text{‘}\textbf{prev}\text{’} & (9) \\
\mathit{binary\_op} & := & \mathit{alu\_op} \ \mid \ \mathit{stream\_op} & (10) \\
\mathit{alu\_op} & := & \text{‘}+\text{’} \ \mid \ \text{‘}-\text{’} \ \mid \ \text{‘}/\text{’} \ \mid \ \text{‘}\times\text{’} \ \mid \ \text{‘}\textbf{and}\text{’} \ \mid \ \text{‘}\textbf{or}\text{’} & (11) \\
\mathit{stream\_op} & := & \text{‘@’} \ \mid \ \text{‘}\textbf{fby}\text{’} & (12) \\
\mathit{func\_call} & := & \mathit{function}\ [\ \text{‘.’}\ \mathit{bound\_dims}\ ]\ \text{‘(’}\ \mathit{bound\_vars}\ \text{‘)’} & (13) \\
\mathit{bound\_dims} & := & \mathit{dims\_list} & (14) \\
\mathit{bound\_vars} & := & \mathit{term\_tuple} & (15) \\
\mathit{where\_clause} & := & \mathit{where\_subject}\ [\ \text{‘}\textbf{where}\text{’}\ \mathit{where\_body}\ \text{‘}\textbf{end}\text{’}\ ] & (16) \\
\mathit{where\_subject} & := & [\ \text{‘\{’}\ ]\ \mathit{term\_tuple}\ \ [\ \text{‘\}’}\ ] & (17) \\
\mathit{where\_body} & := & [\ \mathit{dims\_decls}\ ]\ [\ \mathit{stream\_decls}\ ] & \\
 & & \quad [\ \mathit{func\_decls}\ ]\ \mathit{definition\_list} & (18) \\
\mathit{dims\_decls} & := & \text{‘}\textbf{dimension}\text{’}\ \mathit{dims\_list}\ \text{‘;’} & (19) \\
\mathit{definition\_list} & := & \mathit{definition}\ [\ \mathit{definition\_list}\ ] & (20) \\
\mathit{definition} & := & \mathit{graph\_def} \ \mid \ \mathit{stream\_def} & (21) \\
\mathit{rhs} & := & \mathit{where\_clause} \ \mid \ [\ \text{‘\{’}\ ]\ \mathit{term\_tuple}\ [\ \text{‘\}’}\ ] & (22) \\
\mathit{function} & := & \mathit{graph} \ \mid \ \mathit{seq\_function} & (23) \\
\mathit{graph} & := & \mathit{identifier} & (24) \\
\mathit{seq\_function} & := & \mathit{identifier} & (25) \\
\mathit{graph\_def} & := & \mathit{graph}\ [\ \text{‘.’}\ \mathit{formal\_dims}\ ]\ [\ \text{‘(’}\ \mathit{formal\_vars}\ \text{‘)’}\ ] & \\
 & & \quad \text{‘=’}\ \mathit{rhs}\ \text{‘;’} & (26) \\
\mathit{stream\_def} & := & \mathit{lhs}\ [\ \text{‘.’}\ \mathit{formal\_dims}\ ]\ \text{‘=’}\ \mathit{rhs}\ \text{‘;’} & (27) \\
\mathit{lhs} & := & [\ \text{‘\{’}\ ]\ (\ \mathit{stream} \ \mid \ \text{‘}\textbf{void}\text{’}\ )\ [\ \text{‘,’}\ \mathit{lhs}\ ]\ [\ \text{‘\}’}\ ] & (28) \\
\mathit{cond\_list} & := & \text{‘(’}\ \text{‘(’}\ \mathit{cond\_expr}\ \text{‘)’}\ \mathit{term}\ \text{‘)’}\ [\ \mathit{cond\_list}\ ] & (29) \\
\mathit{cond\_expr} & := & \mathit{cond\_param}\ \mathit{cond\_op}\ \mathit{cond\_param} & (30) \\
\mathit{cond\_param} & := & \mathit{stream} \ \mid \ \mathit{constant} & (31) \\
\mathit{cond\_op} & := & \text{‘==’} \ \mid \ \text{‘! =’} \ \mid \ \text{‘<’} \ \mid \ \text{‘<=’} \ \mid \ \text{‘>’} \ \mid \ \text{‘>=’} & (32)
\end{array}
$$

Figure 1: Q Language in BNF (partial)

$$
\begin{array}{rcll}
\textit{stream\_decls} & := & \textit{stream\_def}\ [\ \textit{stream\_decls}\ ] & (33) \\
\textit{stream\_def} & := & \textit{data\_type}\ \textit{stream}\ \text{`.'}\ \textit{dims\_list} & \\
 & & \quad [\ \text{`='}\ \textit{stream\_init}\ ]\ \text{`;'} & (34) \\
\textit{data\_type} & := & \text{`}\mathbf{int8}\text{'} \mid \text{`}\mathbf{uint8}\text{'} \mid \text{`}\mathbf{int16}\text{'} \mid \text{`}\mathbf{uint16}\text{'} \mid & \\
 & & \text{`}\mathbf{int32}\text{'} \mid \text{`}\mathbf{uint32}\text{'} \mid \text{`}\mathbf{int64}\text{'} \mid \text{`}\mathbf{uint64}\text{'} \mid & \\
 & & \text{`}\mathbf{float}\text{'} \mid \text{`}\mathbf{double}\text{'} \mid \text{`}\mathbf{boolean}\text{'} & (35) \\
\textit{stream\_init} & := & \textit{constant} \mid \textit{init\_list} & (36) \\
\textit{init\_list} & := & \textit{init\_frag}\ [\ \text{`,'}\ \textit{init\_list}\ ] & (37) \\
\textit{init\_frag} & := & \text{`\{'}\ \textit{init\_desc}\ \ \textit{data\_desc}\ \text{`\{'}\ \textit{const\_list}\ \text{`\}\,\}'} & (38) \\
\textit{init\_desc} & := & \text{`\{'}\ \textit{ortho\_desc}\ \text{`\}'} & (39) \\
\textit{data\_desc} & := & \text{`\{'}\ \textit{ortho\_desc}\ \text{`\}'} & (40) \\
\textit{const\_list} & := & \textit{constant}\ [\ \text{`,'}\ \textit{const\_list}\ ] & (41) \\
\textit{ortho\_desc} & := & \textit{ortho\_dim}\ [\ \text{`,'}\ \textit{ortho\_desc}\ ] & (42) \\
\textit{ortho\_dim} & := & \textit{dimension}\ [\ \text{`='}\ \textit{offset} & \\
 & & \quad [\ \text{`+'}\ \textit{extent}\ [\ \text{`/'}\ \textit{sample\_step}\ ]\ ]\ ] & (43) \\
\textit{offset} & := & \textit{constant} & (44) \\
\textit{extent} & := & \textit{constant} & (45) \\
\textit{sample\_step} & := & \textit{constant} & (46)
\end{array}
$$

Figure 2: Q Language Stream Declarations BNF

An atom is either a *constant*, a *stream*, or a *task* — a *func_call* or *operation*. All atoms have a "number of dimensions", which may be zero or more. *func_call* and *operation* also have a "number of arguments" of one or more. Note that in a multidimensional language such as this one, constants may vary over one or more dimensions. Atoms may be optionally enclosed in parentheses to ensure proper interpretation.

The '**if**' operation is redundant, but syntactically convenient. The entire set of *alu_op* operators and some of the *unary_op* ('**-**' and '**not**') are provided mainly for semantic completeness. While supported by an integral basic stream library, they are implemented as functions. It is hoped that Q programs will use a more complex *seq_function*.

Comments may be embedded in the Q language by preceding them by a double slash ('//').

2.1 Stream Declarations

Although it may be possible to determine the characteristics of a stream by examination of an algorithm, the Q language currently requires that streams be declared, along with their data type and dimensionality. The BNF declarations for their definition are in Fig. 2.

$$
\begin{array}{rcll}
\textit{func\_decls} & := & [\ \textit{access\_decls}\ ]\ \textit{func\_prototype}\ [\ \textit{func\_decls}\ ] & (47) \\
\textit{func\_prototype} & := & [\ \text{`\{'}\ ]\ \textit{out\_params}\ [\ \text{`\}'}\ ]\textit{seq\_function} & \\
 & & \quad [\ \text{`.'}\ \textit{dims\_list}\ ]\ \text{`('}\ \textit{in\_params}\ \text{`);'} & (48) \\
\textit{out\_params} & := & \textit{param\_list} & (49) \\
\textit{in\_params} & := & \textit{param\_list} & (50) \\
\textit{param\_list} & := & \textit{data\_type}\ [\ \textit{access\_patt}\ ]\ [\ \text{`,'}\ \textit{param\_list}\ ] & (51) \\
\textit{access\_patt} & := & \textit{identifier} & (52) \\
\textit{access\_decls} & := & \textit{access\_def}\ [\ \textit{access\_decls}\ ] & (53) \\
\textit{access\_def} & := & \text{`}\mathbf{access\_pattern}\text{'}\ \textit{access\_patt} & \\
 & & \quad \text{`\{'}\ \textit{access\_flags}\ \text{`,'}\ \textit{dims\_desc}\ \text{`\};'} & (54) \\
\textit{access\_flags} & := & \text{`}\mathbf{fixed}\text{'}\ |\ \text{`}\mathbf{variable}\text{'} & (55) \\
\textit{dims\_desc} & := & \text{`\{'}\ \textit{ortho\_dim}\ \text{`,'}\ \textit{func\_step}\ \text{`,'}\ \textit{dimension}\ \text{`\}'} & \\
 & & \quad [\ \text{`,'}\ \textit{dims\_desc}\ ] & (56) \\
\textit{func\_step} & := & \textit{constant} & (57)
\end{array}
$$

Figure 3: BNF for Q Language Function Declarations

2.2 Sequential Function Declarations

If a *function* is represented by a *seq_function* (a Q low level function), a description of the function's input and output parameters must be provided (in a manner similar to the function prototypes of the ANSI C programming language). The syntax of the function declations is specified by the BNF definitions in Fig. 3.

If no access pattern is specified for a function input or output, a scalar access is assumed.

2.3 Intensional Primitives

Two elements of the Q language are not directly translatable into a Q dataflow level algorithm representation, yet understanding them is crucial to understanding the language. They are the direct addressing (or permutation) operator ('@') and the context query operator ('#', really a constant). They are included in the language as primitives, and are used (for example) describe the '**resample**' and '**upsample**' stream operators.

Aside from the typical numeric constants, the Q language contains an "indexing" constant ('#'), which is monadic (it varies along one dimension) and specifies the location along the bound dimension at which evaluation is taking place.

The direct addressing operator ('@') allows access to a stream element of a stream at a particular location (specified absolutely) along a specified dimension. It is monadic and binary. In its general form, it allows data-dependent access to any element in a stream and is difficult to support. More limited versions are provided for program use: *e.g.* '**first**', '**next**', '**prev**', '**resample**' and '**upsample**'.

2.4 Stream Operators

The '**resample**' and '**upsample**' stream operators are monadic intensional operators, represented as low level functions in the language description. They must be implemented as operators, however, as they are polymorphic and have an access pattern on their first input argument which is not constant (although fixed in extent.) A pseudo-prototype for these functions is given below, with polymorphic arguments typed as '**anytype**', and the non-constant access patterns represented by '**resample_patt**' or '**upsample_patt**'.

```
anytype resample.d( anytype resample_patt, int32, int32 );
anytype upsample.d( anytype upsample_patt, int32, int32, int8 );
```

A more complicated stream operator, '**select**', is also represented as a low level function. Although polymorphic, this operator has fixed, constant, and mostly scalar access patterns. It supports several forms, each with slightly different semantics. The semantics of the stream operators are described in the *Q Instruction Set Architecture.*

```
access_pattern select_range { 1, fixed, { 0, 3, 1, 0, 0 } }

anytype select.d( anytype, int32, int32, int32 );
anytype select.d,r( anytype, int32 select_range );
anytype select.d,t( anytype, int32, int32, int32 );
anytype select.d,t,r( anytype, int32 select_range );
```

3 The Syntax of Multidimensional Lucid

The BNF description of the Lucid language is presented in Figure 4. It was derived from the description presented in Ch. 3 of *Multidimensional Programming*, with a the Lucid *variable* is called a *stream.*

A term N-tuple is a list of N terms. If N is greater than one, the curly braces ({ }) are required around the entire list.

All atoms have both a "number of dimensions" and a "number of arguments". The atoms are slightly redundant, as a *stream* is a *function call* with no (zero) arguments and a *constant* is an *operation* with no arguments. The Lucid language contains a monadic context query constant ('#'), identical to the one in the Q language.

$$
\begin{array}{rcll}
\textit{program} & := & \textit{term} & (1) \\
\textit{term} & := & \textit{where\_clause\_of\_size\_1} \;\mid\; \textit{atom} & (2) \\
\textit{atom} & := & \textit{constant} \;\mid\; \textit{stream} \;\mid\; \textit{operation} \;\mid\; \textit{func\_call} & (3) \\
\textit{term\_tuple} & := & \textit{term} \; [\text{ `,' } \textit{term\_tuple} \;] & (4) \\
\textit{stream} & := & \textit{identifier} & (5) \\
\textit{operation} & := & (\; \textit{unary\_op} \; [\text{ `.' } \textit{dims\_list} \;] \; \textit{term} \;) \mid & \\
 & & (\; \textit{term} \; \textit{binary\_op} \; [\text{ `.' } \textit{dims\_list} \;] \; \textit{term} \;) \mid & \\
 & & (\text{ `}\textbf{if}\text{' } \textit{term} \text{ `}\textbf{then}\text{' } \textit{term} \; [\text{ `}\textbf{else}\text{' } \textit{term} \;] \text{ `}\textbf{fi}\text{' }) & (6) \\
\textit{dims\_list} & := & \textit{dimension} \; [\text{ `,' } \textit{dims\_list} \;] & (7) \\
\textit{dimension} & := & \textit{simple\_identifier} & (8) \\
\textit{unary\_op} & := & \text{`}-\text{'} \mid \text{`}\textbf{not}\text{'} \mid \text{`}\textbf{first}\text{'} \mid \text{`}\textbf{next}\text{'} & (9) \\
\textit{binary\_op} & := & \textit{alu\_op} \;\mid\; \textit{intensional\_op} & (10) \\
\textit{alu\_op} & := & \text{`}+\text{'} \mid \text{`}-\text{'} \mid \text{`}/\text{'} \mid \text{`}\times\text{'} \mid \text{`}\textbf{and}\text{'} \mid \text{`}\textbf{or}\text{'} & (11) \\
\textit{intensional\_op} & := & \text{`@'} \mid \text{`}\textbf{fby}\text{'} \mid \text{`}\textbf{asa}\text{'} \mid \text{`}\textbf{wvr}\text{'} \mid \text{`}\textbf{upon}\text{'} & (12) \\
\textit{func\_call} & := & \textit{function} \; [\text{ `.' } \textit{bound\_dims} \;] \text{ `(' } \textit{bound\_vars} \text{ `)'} & (13) \\
\textit{bound\_dims} & := & \textit{dims\_list} & (14) \\
\textit{bound\_vars} & := & \textit{term\_tuple} & (15) \\
\textit{where\_clause} & := & \textit{where\_subject} & \\
 & & \quad [\text{ `}\textbf{where}\text{' } \textit{where\_body} \text{ `}\textbf{end}\text{' }] & (16) \\
\textit{where\_subject} & := & [\text{ `\{' }] \; \textit{term\_tuple} \; [\text{ `\}' }] & (17) \\
\textit{where\_body} & := & [\; \textit{dims\_decls} \;] \; \textit{definition\_list} & (18) \\
\textit{dims\_decls} & := & \text{`}\textbf{dimension}\text{' } \textit{dims\_list} \text{ `;'} & (19) \\
\textit{definition\_list} & := & \textit{definition} \; [\; \textit{definition\_list} \;] & (20) \\
\textit{definition} & := & \textit{lhs} \text{ `=' } \textit{rhs} \text{ `;'} & (21) \\
\textit{rhs} & := & \textit{where\_clause} \;\mid\; [\text{ `\{' }] \; \textit{term\_tuple} \; [\text{ `\}' }] & (22) \\
\textit{function} & := & \textit{identifier} & (23) \\
\textit{lhs} & := & [\text{ `\{' }] \; \textit{lhs\_list} \; [\text{ `\}' }] \; [\text{ `.' } \textit{formal\_dims} \;] & \\
 & & \quad \text{`(' } \textit{formal\_vars} \text{ `)'} & (24) \\
\textit{lhs\_list} & := & (\; \textit{function} \;\mid\; \textit{stream} \;) \; [\text{ `,' } \textit{lhs\_list} \;] & (25)
\end{array}
$$

Figure 4: BNF description of Lucid