Flow Structure and Completeness

Karl Fant

April, 2015

Flow Tokens

Tokens declare units of flow completeness.

- There are no predefined tokens.
  - Tokens are defined from scratch as a hierarchy of name relations.
- The leaves of the hierarchy - the terminal names - represent single rails.

The hierarchy is structured in terms of AND-OR relations.

- The AND relation indicates tokens that must all be present for completeness.
- The OR relation indicates tokens only one of which must be present for completeness.
- In a token declaration brackets [ ] encompass AND related tokens and braces { } encompass OR related tokens.

A range can be specified for token names

- [0-9] specifies AND related names 0, 1 ... 8, 9
- {B-F} specifies OR related names B, C ... E, F
- [bit(S-Z)] specifies AND related names bitS, bitT ... bitY, bitZ
- {bit(0-9)} specifies OR related names bit0, bit1 ... bit8, bit9
- A name range may imply ordering but does not specify an ordering.
  - Ordering arises from the flow of reference to the names.
  - How the carry associates from name to name in a numeric addition for instance.

The token structure is just a name referencing scheme that includes a completeness property in the AND-OR relations.

Tokens with numeric values and structures

Binary digit

token bit{1,0}; //declares a two rail token named bit with OR related rail names 0,1

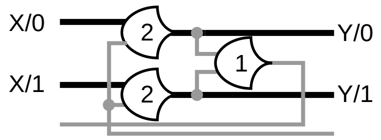
bit/0 ... bit/1

token (X, Y)[bit]; //declares X and Y as two rail tokens with OR related rail names 0,1. previously declared bit is used as a template

token X{1,0}, Y{1,0};

X/0 ... Y/1

token bit{1,0};
token (X, Y)[bit];
(X -> Y)



completeness of Y = #Y = Y/0 | Y/1;

Binary number

token byte[0-7][bit]; token byte[0-7]{0,1};

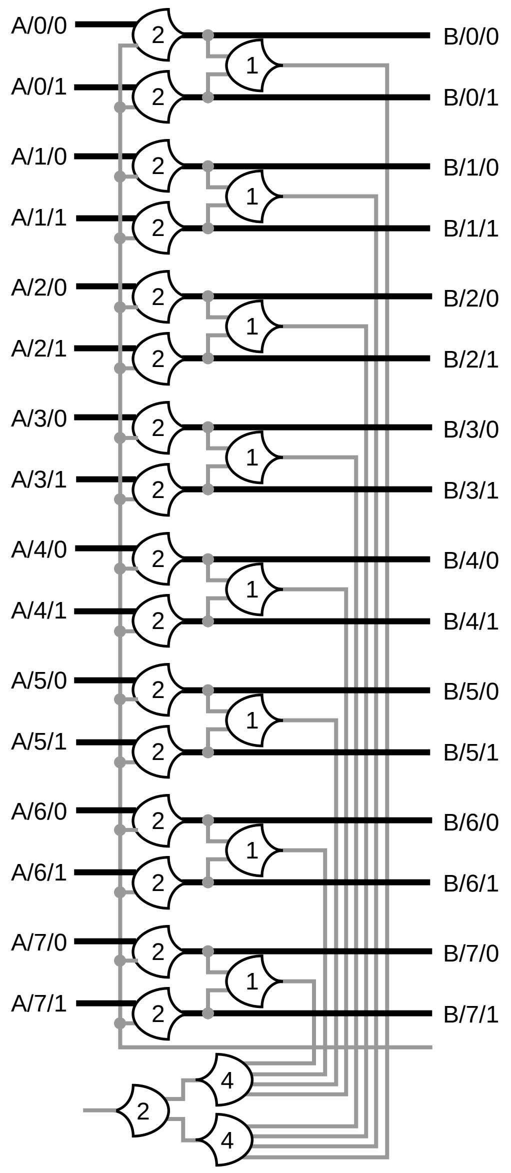
byte/0/1 ... byte/1/0 ... byte/7/1 ... byte/5/0

token (A, B, C)[byte];

A/5/0 //refers to rail named 0 of digit 5 of A
B // refers to the entire structure of B
A/4 // refers to digit 4 of A

token bit{1,0};
token byte[bit][0-7];
token (A, B, C)[byte];

(A -> B)



completeness of B = #B = (B/0/0 | B/0/1) & (B/1/0 | B/1/1) & (B/2/0 | B/2/1) & (B/3/0 | B/3/1) & (B/4/0 | B/4/1) & (B/5/0 | B/5/1) & (B/6/0 | B/6/1) & (B/7/0 | B/7/1);

Tokens with numeric values and structures

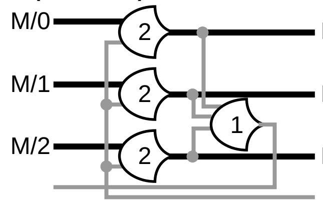
Trinary
token trit{2-0};

trit/0 ... trit/1 ... trit/2

token (M,N)[trit];

M/0 ... M/1 ... N/2 ... N/3

token trit{2-0};
token (M, N)[trit];
(M -> N);



completeness of N = #N = N/0 | N/1 | N/2;

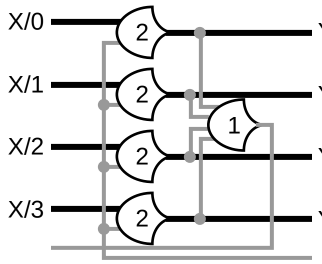
Quaternary
token quat{3-0};

quat/0 ... quat/1 ... quat/2 ... quat/3

token (X,Y)[quat];

X/0 ... X/1 ... Y/2 ... Y/3

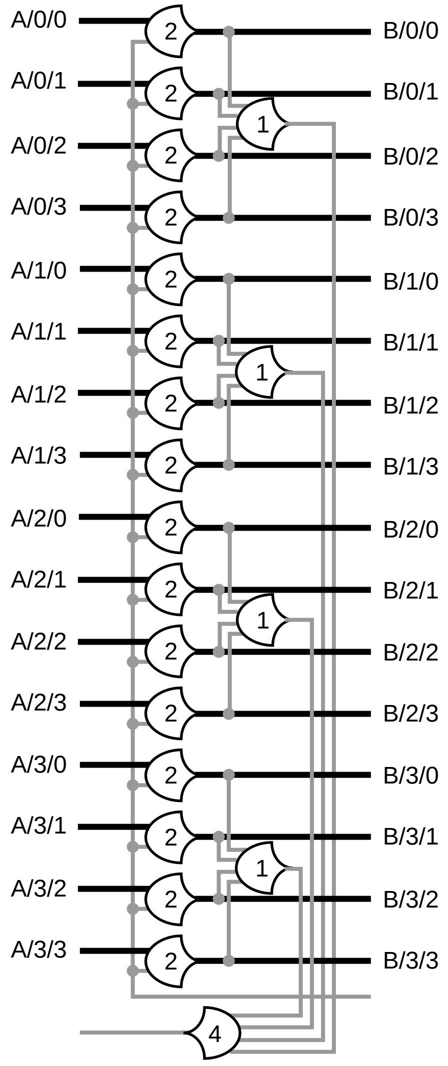
token quat{3-0};
token (X,Y)[quat];
(X -> Y);



completeness of Y = #Y = Y/0 | Y/1 | Y/2 | Y/3;

Decimal
token dit{9-0};
dit/0 ... det/6 ... det/9
token decimal[0-9][dit];
decimal/0/1 ... decimal/1/3 ... decimal/7/9 ... decimal/5/0

token quat{3-0};
token qbyte[0-3][quat];
token (A,B)[qbyte];
(A -> B)

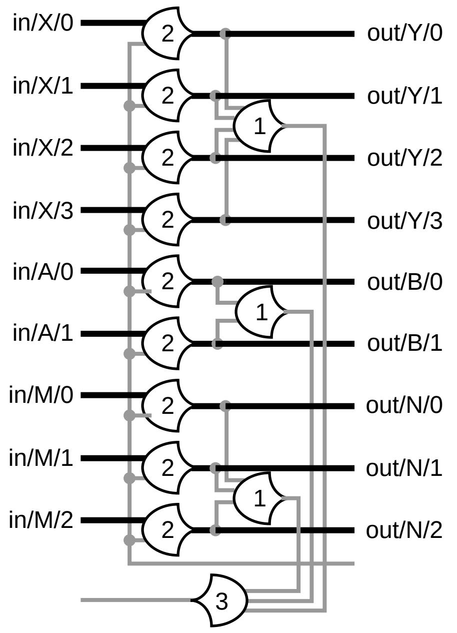


```
A/2/3  // references rail named 3 of quaternary digit 2 of A
A      // refers to the entire structure
B/3    // refers to digit 3 of B
```

completeness of B = #B = (B/0/0 | B/0/1 | B/0/2 | B/0/3) & (B/1/0 | B/1/1 | B/1/2 | B/1/2)
& (B/2/0 | B/2/1 | B/2/2 | B/2/3) & (B/3/0 | B/3/1 | B/3/2 | B/3/3);

Tokens with heterogeneous numeric values and structures

```
token bit{1,0};
token (A, B)[bit];
token trit{2-0};
token (M,N)[trit];
token quat{3-0};
token (X,Y)[quat];
token in[X, A, M];
token out[Y, B, N];
(in -> out);
```



completeness of out = #out = (out/Y/0 | out/Y/1 | out/Y/2 | out/Y/3) & (out/B/0 | out/B/1) & (out/N/0 | out/N/1 | out/N/2);

Floating point

```
token bit{0,1};
token sign[bit];
token mantissa[0-22][bit];
token exponent[0-7][bit];
token float[sign, exponent, mantissa];
token (A, B, C)[float];
```

references

```
        A
        A/sign
        C/mantissa/21
        B/mantissa/21/0
        C/exponent/4/1
```

Non numeric tokens

Kitchen measures

```
token tbsp{empty,full};
token cup[tbsp(1-16)][tbsp];
token quart[cup(0-7)][cup];
token (A, B, C)[quart];

references
          A/cup5/tbsp1/full      //refers to the full rail of tbsp 1 of cup 5 of quart A
          A                      // refers to quart A
          B/cup4                 // refers to cup 4 of B
          C/cup7/tbsp5           //refers to tbsp 5 of cup 7 of quart C
```

Categories

```
token family{mother,father, sister, brother};
token (Smith, Jones, Hitchkock)[family];

references
          Smith/mother ... Jones/brother
```

Records and Emptyness

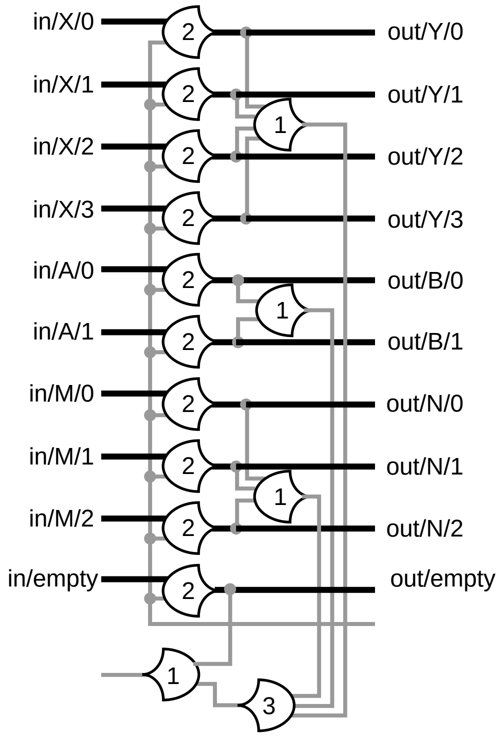
A flow structure that may have a value or be empty such as a field of a data base record can contains a single rail indicating empty as well as the structure that may contain content.

So there is another level of OR relation to the flow structure.

Empty functions like zero as a place holder in a structure meaning this place is empty.

The flowing the record is empty or it has content.

```
token empty;
token bit{1,0};
token (A, B)[bit];
token trit{2-0};
token (M,N)[trit];
token quat{3-0};
token (X,Y)[quat];
token in{[empty], [X, A, M]}
token out{[empty], [Y, B, N]};
(in -> out);
```



completeness of out = #out = ((out/Y/0 | out/Y/1 | out/Y/2 | out/Y/3) & (out/B/0 | out/B/1) & (out/N/0 | out/N/1 | out/N/2)) | out/empty;

The AND-OR Counterpoint

AND relations provide a constant referent structure within which variable OR relations flow forming the essential counterpoint of computation.

Variation in relation to stable referent.

Neither has meaning without the other.

Token declarations are the primitive specification of this AND-OR flow structure.

An 8 bit number always has 8 bits whose values (rails) may vary as they flow through transform functions but there is always a constant 8 bits.