Dependency language:
Supplement to
A Journey through Computation

Karl Fant

The dependency language of computation was introduced in “A Journey Through Computation” and used in the narrative which focused on the behaving networks. This supplement focuses on the rationale and form of the language itself.

1.1. Syntax structure and name correspondence

A dependency language expression is a string of symbols structured with syntax relations and name correspondence relations representing a static network of dependency relations through which flows dynamic transition behavior.

1.1.1. Symbols

A names is expressed with contiguous name forming symbols a A to z Z, 0 to 9, /, “.” and _ delimited by white space, “space”, “CRLF”, and syntax symbols. Syntax structures are expressed with syntax symbols disjoint from the name forming symbols, [ ], { }, ( ), <=, =>, ~, ?, #, -, : and “,” and “.”.

1.1.2. Syntax structure

Syntax structure relations express local, contiguous, directional, one to one and many to one dependency relations but do not express one to many dependency relations. A syntax expression isolates and connects different names associated to the syntax expression.

1.1.3. Name correspondence

Name correspondence relations express remote, noncontiguous, nondirectional one to one and one to many dependency relations but do not express many to one dependency relations. A name correspondence relation connects different isolated syntax expressions associated with the name.

1.1.4. The collaborative interplay

A same name occurring in two or more different syntax expressions dependently associates the syntax expressions through the correspondence of the name. A name correspondence relation does not have an intrinsic structure but derives its structure and directionality from the syntax structures in which the name occurs. One occurrence of a name is a result output of a syntax expression. All other occurrences of the same name are dependent interaction inputs to different syntax expressions.

- Name correspondence dependently connects different isolated syntax structures extending the expressivity of syntax.
- Syntax expression isolates and dependently connects different names extending the expressivity of name correspondence.

The two complementary expressions of dependency relation mutually renew the expressive range of each other enabling indefinitely extendable expressions of dependency relations. There is always a bit of each in every dependency expression. The proportion of syntax expression and name correspondence varies along a tradeoff spectrum. For instance a functional expression is mostly syntax expression and the machine language expression to which the functional expression is compiled is mostly name correspondence.[1]

1.2. Primitivity

The dependency language is presented in the context of association differentiation with the primitive association interaction behaviors **“all of”** and **”one of”** and the primitive differentness conditions **D** and **N**.[2]

1.2.1. Primitive differentness of condition

The primitive differentness condition named **N** explicitly represents “**N**ot an interaction differentness”. The primitive differentness condition named **D** explicitly represents “an interaction **D**ifferentness”. A transition from **N** to **D** represents an appearance of one instance of interaction differentness. A transition from **D** to **N** represents a disappearance of, emptiness of, instance of interaction differentness. By monotonically transitioning between **D** and **N** as in Figure 1.1 **D** and **N** manifest primitive mutually exclusive condition differentiation. Conditions **D** and **N** are mutually exclusively different from each other by virtue of transition. Every transition from **N** to **D** represents an exclusively different instance of **D**.

The monotonic transitioning also expresses primitive discreteness of differentness. The transitioning might be continuous but the appreciation of extremes, with thresholding for instance, is discrete.

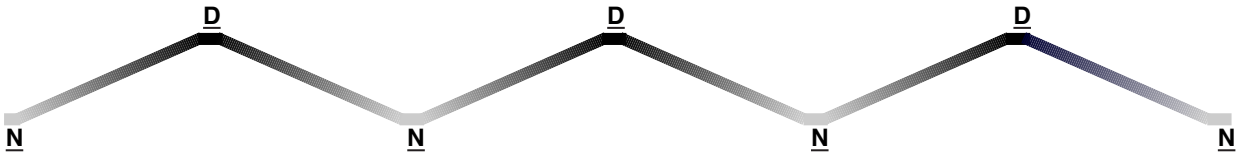


Figure 1.1. Primitive mutually exclusive differentness.

1.2.2. The primitive condition interaction behaviors

The primitive interaction behaviors **“all of”** and **“one of”** are shown in relation to primitive differentness conditions **D** and **N** in Figure 1.2. The tables represent the condition mapping behaviors from interaction input to result output. The “–” means that no result output transition occurs. Enclosing braces **{ }** indicate **“one of”** related behavior for which **D** completeness is one interaction input at **D** and the rest at **N**. Enclosing brackets **[ ]** indicate **“all of”** related behavior for which **D** completeness is all interaction inputs at **D** and none at **N**. Completely **N** is the same for all behaviors with all interaction inputs at **N**, i.e., empty of interaction differentness. The transition of a behavior result output recognizes the transition of its interaction inputs between **D** completeness and completely **N**.

The syntax **<=** or **=>** indicates **“is dependent on”**. Both **A<=B** and **B=>A** indicate that **A** is dependent on **B**. Figure 1.2 includes the graphic representation and the syntactic representation of each primitive interaction behavior.

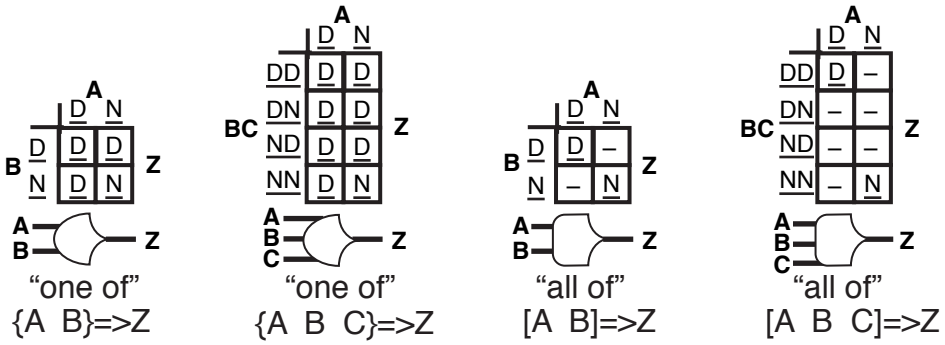


Figure 1.2. Primitive interaction behavior interaction propensities.

1.2.2.1. The "all of" behavior, *mutual inclusivity*

Z<=[A B C] /* Z is dependent on "all of" A, B and C */

If **"all of" A** and **B** and **C** transition from **N** to **D** then **Z** will transition to **D**. When **"all of" A** and **B** and **C** have transitioned to **N** then **Z** will transition to **N**.

The syntax structure **<=[ ]** dependently connects the different names **Z**, **A**, **B** and **C**.

1.2.2.2. The "one of" behavior, *mutual exclusivity*

Z<={A B C} /* Z is dependent on "one of" A, B and C */

If **"one of" A** or **B** or **C** transitions from **N** to **D** then **Z** will transition to **D**. If a second input transitions to **D** then **Z**, already at **D**, does not transition. When **"all of" A** and **B** and **C** have transitioned to **N** then **Z** will transition to **N**.

The syntax structure **<={ }** dependently connects the different names **Z**, **A**, **B** and **C**.

For the rationale underlying the monotonically transitioning differentnesses and the primitive behaviors see "A Journey Through Computation" Chapter 5.

1.2.3. Non interaction primitive behaviors

There are two primitive behaviors that do not participate in the interactions of differentnesses but that participate in the coordination of the dependent flow of interaction behavior.

The first behavior on the left of Figure 1.3 is referred to as **conversion** because it is used in network completeness closure to convert **D** to **N** and **N** to **D**. Conversion only appears in closure and never appears in an interaction dependency relation (see section 1.6). The inverter symbol is still used because it is convenient and should be easily understood in context.

The second behavior on the right of Figure 1.3 is the **mutex/arbiter**, represented syntactically as **{{ }}**, that constrains two otherwise uncoordinated transition flows to flow one at a time, mutually exclusively. In Figure 1.3 **A** flows to **A** and **B** flows to **B** but only **"one at a time"** turning two uncoordinated transition flows into a coherent **"one of"** related flow. The **arbiter** can be viewed as enforcing **"one of"** related behavior (see section 1.10.3).[3]

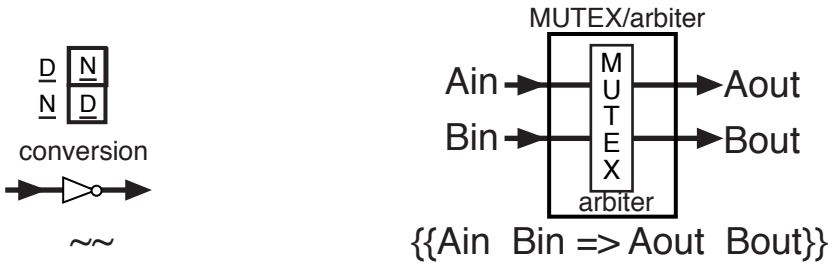


Figure 1.3. Non interacting flow coordination behaviors.

1.2.3.1. Initializing primitive behaviors

The behaviors begin initially empty of interaction differentness with all interaction inputs at **N** and asserted result output at **N**. The interaction inputs can then transition to **D** completeness and continue with monotonic transitioning. Figure 1.4 show the initiaization behaviors.

The single tilde ~ represents a conversion that is forced to **N** during initialization to allow an initial completely **N** wavefront to propagate through a network. The less frequently occurring double tilde ~~ represents the not initialized conversion behavior which is used only with initialization to **D** completeness (see section 1.8.1.1).

The **"all of"** behavior can be initialized to specific result output assertion conditions regardless of the presented interaction input conditions with an explicit init condition. When init is enabled the result is forced to a specific condition. When init is disabled the behavior resumes its intrinsic behavior. The **"all of"** behavior can be initialized to **D** or to **N**. The initialized conversion behavior is always initialized to **N**.

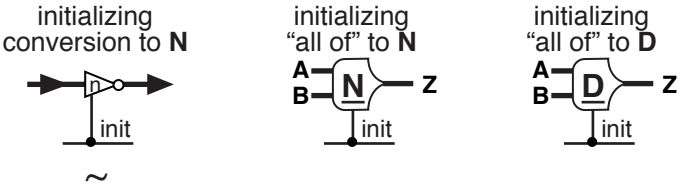


Figure 1.4. Explicitly initializable behaviors.

1.2.4. The differentness of Primitive interaction behavior

Primitive interaction behaviors are differentiated only in terms of the appreciation of their input transition to **D** completeness. Appreciation of the transition of input to completely **N** is universal for all behaviors. The behavior of the primitive interaction behaviors is illustrated in Figure 1.5.

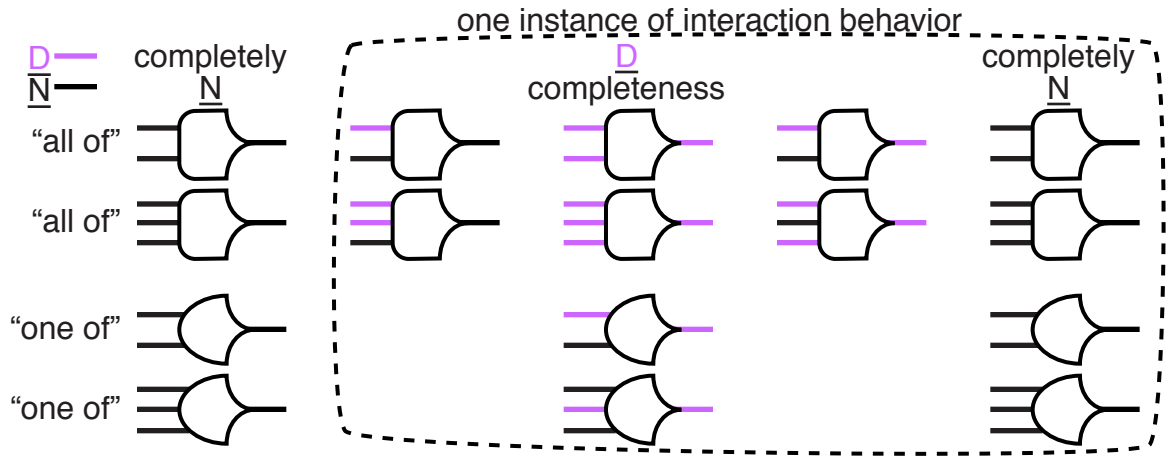


Figure 1.5. primitive behaviors with input completeness transition behavior.

1.2.4.1. The primitive completeness criterion

A primitive interaction behavior begins empty of interaction differentness with interaction inputs completely **N** and its result output asserting **N**. When the interaction inputs transition to **D** completeness the result output transitions to **D** which is maintained until the interaction inputs transition to completely **N** at which point the result output transitions to **N** which is maintained until the interaction inputs transition to **D** completeness at which point the result output transitions to **D** which is maintained until the interaction inputs transition to completely **N** and so on.

The transition of the result output recognizing the transition of the interaction inputs to completeness is the primitive completeness criterion.

1.2.4.2. D completeness and completely N

The primitive behaviors establish the monotonic transition behavior between **D** completeness and completely **N**, illustrated in Figure 1.6,.

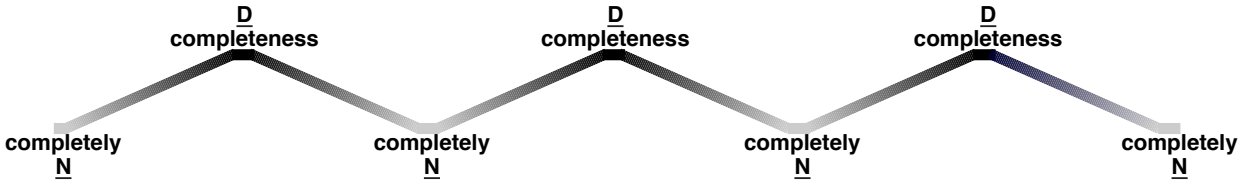


Figure 1.6. Monotonically transitioning completeness relations.[4]

1.2.5. The confluence of primitivity

A primitive interaction behavior establishes primitive dependency relations as well as the primitive directionality of dependency. The relative persistence of a primitive behavior also establishes primitive sameness of place while its isolated interaction inputs and result output establish primitive differentness of place each of which represents a sameness of place of monotonic transitioning between **D** and **N** which establishes primitive differentness of condition

and primitive differentness of time. Each transition from **N** to **D** marks a differentness of interaction condition in successive differentnesses of time at a sameness of place.

The transition of the interaction places of a same primitive behavior to **D** completeness and the transition of its result place to **D** followed by the transition of its interaction places to completely **N** and the transition of its result place to completely **N** represents one primitive directional interaction of the differentness conditions of two different and dependent interaction places to one different result place through one primitive sameness of interaction behavior in one primitive differentness of time.

It is this counterpoint between sameness of place and differentness of place, between differentness of dynamic transitioning through sameness of static interaction that engenders computation.

1.2.6. Constancy and composition

The behavior of each primitive interaction behavior is constant in that it always delivers the same result output condition for the same presented completeness of interaction input conditions which allows the primitive behaviors to be indiscriminately referenced, replicated and copied to anywhere and anywhen to, in particular, express networks of primitive behaviors.

Expression of the dependent interaction of differentness is extended with primitive interaction behaviors associated result output to interaction input forming a static **network of dependency relations** such as shown in Figure 1.7. While each primitive behavior is different in the network each association relation between the behaviors representing an interacting differentness occupies a uniquely different **place of association** within the network. These between places of association can be named and their interaction expressed in terms of the primitive behaviors.

The expression:

[A B]=>M
[A C]=>N
[B C]=>O
{M N O}=>Z

Expresses the network of Figure 1.7.

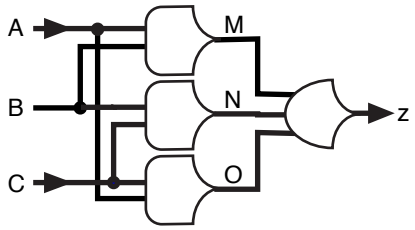


Figure 1.7. Primitive composition of primitive interaction behaviors.

In the first three syntax structures of the expression above the sameness of each syntax structure associates three different names. In the fourth syntax structure the sameness

(correspondence) of the names **M**, **N** and **O** associate the first three different syntax structures to the fourth different syntax structure expressing the dependence of **Z** on **A**, **B** and **C**. In the expression below the four different names **A**, **B**, **C** and **Z** in the sameness of a single syntax structure with syntactic nesting relations.

{ [A B] [A C] [B C] }=>Z

Notice that **M**, **N** and **O** are no longer referenced as their dependency relations are now expressed syntactically illustrating the tradeoff between expression with syntax structure relation and name correspondence relation.

The above expressions and network associate and interacts single places of association each of which monotonically transitions between **D** and **N**. If any two input places of association transition to **D** the result place of association **Z** will transition to **D**.

1.3. The first level locality of interaction differentness

A locality represents mutually exclusive interaction differentnesses beyond the single place of association differentness. A first level locality consists of two or more **"one of"** related places of association with each place representing one differentness of the locality. The **"one of"** relation means that **D** completeness for the locality is the transition of one place of association to **D** while the rest remain at **N** expressing one differentness condition of the locality in one differentness of time at one sameness of place. The locality as a whole is the sameness of place.

The mutually exclusive differentness represented by a locality is a collaborative confluence of three differentnesses. Each successive transition to **D** completeness wavefront flowing through a locality transiently asserts one of the locality's mutually exclusive differentnesses at the locality's persistent differentness of place in a network in a recurring differentness of time.

1.3.1.1. Expressing a first level locality

A first level locality is named and its differentnesses are named. A locality named **A** containing two **"one of"** related places of association named **1** and **0** is expressed as:

A/{1 0}/{D N}

The slash / indicates a withiness relation. The braces **{ }** indicate **"one of"** related. The expression can be read as **A** is dependent on **"one of" 1** or **0** each of which is dependent on **"one of" D** or **N**. The above expression expands to:

```
A/{1/{D N}  0/{D N}}          /* expands to */
{A/1/{D N}  A/0/{D N}}
```

"one of" 1 or **0** is within **A**. **"one of" D** or **N** is within each of **1** and **0**. Since every place of association can only assert **D** or **N** the **{D N}** term is a universal most primitive terminal and can be implied with a terminal dangling slash.

```
A/{1 0}/                  /* expands to */
A/{1 0}/{D N}
```


The differentness names **1** and **0** of the locality are isolated within the locality and can be used over in different localities. Three localities **A**, **B** and **C** which all have the same locality structure can be expressed as a list of unrelated names aggregated by parentheses **( )** followed by a term that distributes to each name.

(A B C)/{1 0}/ /* expands to */
A/{1 0}/ B/{1 0}/ C/{1 0}/ /* and so on */

1.3.1.2. Referencing a first level locality

A first level locality differentness is referenced with a pathname to each differentness place of association formed by distributing the locality name over its differentness names forming all the possible pathname references to the locality differentnesses.

A/{1 0} B/{1 0} C/{1 0} /* distribute to */

{A/1 A/0} {B/1 B/0} {C/1 C/0}

Each reference to a locality differentness is a query as to whether the differentness is **D** or **N**. The **D** or **N** is not included in a reference pathname and the terminal dangling slash is not present.

A/1 A/0 B/1 B/0 C/1 C/0 /* and so on */

A locality as a whole is referenced with the locality name which queries the locality as a whole.

A B C

1.3.1.3. Expressional generality of first level locality

A first level locality can be expressed with arbitrarily chosen names representing any number of places of association.

E/{2 1 0}/ G/{B 4 p 7}/ hexadecimal/{F E D C B A 9 8 7 6 5 4 3 2 1 0}/
familymember/{mother father daughter son}/

Locality **G** above expands to:

G/{B 4 p 7}/{D N} **/* then to */**
G/{B/{D N} 4/{D N} p/{D N} 7/{D N}} **/* then to */**
{G/B/{D N} G/4/{D N} G/p/{D N} G/7/{D N}} **/* and so on */**

The differentnesses of locality **G** are referenced as:

G/B G/4 G/p and G/7

Second level localities which are compositions of first level localities are introduced later in section 1.11.

1.4. Expressing first level locality Interaction

The dependent interaction of first level localities is expressed by associating locality differentness pathname references within the primitive interaction behavior references.

1.4.1. "all of" behavior

The expression **[A/0 B/1]** referencing the locality differentnesses **A/0** and **B/1** in an **"all of"** relation means that if **"all of" A/0** and **B/1** are **D** then the **"all of"** relation transitions to **D**. When both **A/0** and **B/1** transition to **N** the **"all of"** relation transitions to **N**.

1.4.2. "one of" behavior

The expression **{A/0 B/1}** referencing the locality differentnesses **A/0** and **B/1** in an **"one of"** relation means that if **"one of" A/0** or **B/1** are **D** then the **"one of"** relation transitions to **D**. When both **A/0** and **B/1** are transitioned to **N** the **"one of"** relation transitions to **N**.

1.4.3. Syntactic behavior nesting

With **{[A/0 B/1] [A/1 B/0]}** if **[A/0 B/1]** or **[A/1 B/0]** transition to **D** then **{[A/0 B/1] [A/1 B/0]}** transitions to **D**.

1.4.4. Locality nesting

Locality nesting is expressed by recapitulating the expression of the result locality and nesting within the recapitulated expression the dependency relation, **<=**, for each of its differentnesses.

For locality **C/{1 0}/** if **C/1<={[A/0 B/1] [A/1 B/0]}** and **C/0<={[A/0 B/0] [A/1 B/1]}** its locality nested expression is

```
C/{1<={[A/0 B/1] [A/1 B/0]}
   0<={[A/0 B/0] [A/1 B/1]} }
```

Locality **C** is dependent on localities **A** and **B**.

1.4.5. The dependency expression:

The expression:

```
(A/{1 0}/  B/{1 0}/  C/{1 0}/)          /*  locality expressions  */
     /*  dependency relations  */
     C/{1<={[A/0 B/1] [A/1 B/0]}
        0<={[A/0 B/0] [A/1 B/1]} }
```

expresses the network:

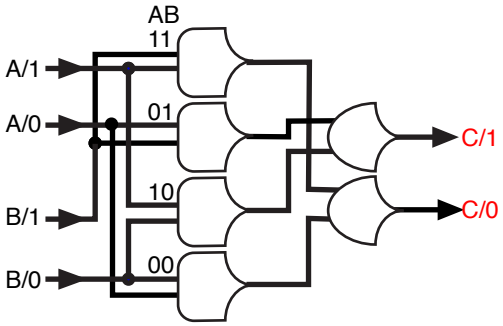


Figure 1.8. A first level locality interaction network.

Locality **A** representing mutually exclusive differentness and locality **B** representing mutually exclusive differentness mutually inclusively associate to interact through a network of primitive interaction behaviors and produce a mutually exclusive differentness of locality **C**. Only one of the locality nested terms will resolve to **D** and transition one of the differentnesses of **C** to **D**. The mutual exclusivity of the terms traces back to the mutually exclusivity of the differentnesses of locality **A** and the mutual exclusivity of the differentnesses of locality **B**.

1.4.6. Recognizing the interacting differentnesses

Each interacting locality represents a range of mutually exclusive differentnesses. Cross associating all of the differentnesses of each locality results in unique composite path names of all the possible interactions of the localities, **A/0 B/1**, **A/1 B/0**, **A/1 B/1** and **A/0 B/0**. Each interacting locality transitions one of its places of association to **D** and only one of the cross associations will present **DD** while the remaining cross associations will present only one **D** or no **D**. The interacting differentnesses can be recognized by appreciating the **DD** cross association with **"all of"** behavior terms, **[A/0 B/1]**, **[A/1 B/0]**, **[A/1 B/1]** and **[A/0 B/0]**. One of the **"all of"** behaviors appreciates the one **DD** by transitioning its result to **D** initiating a path of transition through the dependency relations that transitions the result locality to **D** completeness. When the interacting localities transition to completely **N** the **DD** presentation will transition to **NN** and the recognizing **"all of"** behavior will transition its output to **N** initiating a path of transition through the dependency relations that transitions the result locality to completely **N**.

1.4.7. The dynamic assumption

At this point a network expression assumes that its interacting localities will monotonically and simultaneously transition between **D** completeness and completely **N** and that each transition will be held long enough for the network result locality to transition in response. This behavior is provided by an agency extrinsic to the expression which must appreciate the delay relations of the expression in relation to the extrinsic agency's own time metric. The dependency language will ultimately transcend this extrinsic dependency but the development of the language has to pass through this stage of extrinsic dependency.

1.4.8. Initialization

The initial presentation to a network must be completely **N** which will transition all of the behaviors in the network to **N** emptying the network of interaction differentness. If the initial

presentation is **D** completeness there could have been **D** conditions in the network and the result locality of the network could ambiguously glitch. While the next presentation to completely **N** would initialize the network an initial presentation of **D** completeness cannot be guaranteed to work.

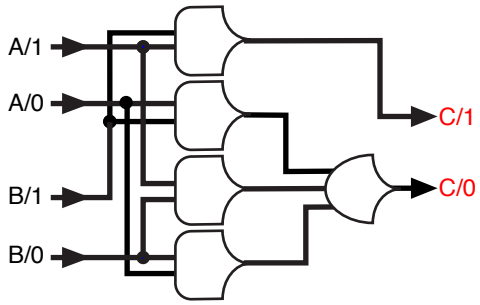
1.4.9. The completeness criterion

The result locality **C** transitions to **D** completeness only when the interacting localities transition to **D** completeness and transitions to completely **N** only when the interacting localities transition to completely **N**. The transition of the result locality to completeness implies the transition of the interacting localities to completeness fulfilling the completeness criterion.

There can be expressions that do not fulfill the completeness criterion.

Fulfills the completeness criterion

```
/*  localities  */
 (A/{1 0}/   B/{1 0}/   C/{1 0}/)
        /*  dependency relations  */
        C/{1<=[A/1 B/1]
           0<={[A/0 B/0] [A/0 B/1] [A/1 B/0] } }
```



Does not fulfill the completeness criterion/

```
*  localities  */
 (A/{1 0}/   B/{1 0}/   C/{1 0}/)
        /*  dependency relations  */
        C/{1<=[A/1 B/1]
           0<={A/0 B/0} }
```

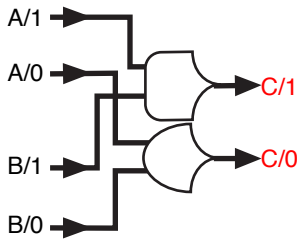


Figure 1.9. Fulfilling and not fulfilling the completeness criterion

The upper expression and network of Figure 1.9 fulfills the completeness criterion. The lower expression and network performs the same interaction mapping behavior asserting the correct result but it does not fulfill the completeness criterion. The the result locality differentness **C/0** can transition to **D** with just one interaction locality transitioned to **D** completeness. This narrative considers only expressions that fulfill the completeness criterion.

1.4.10. Name correspondence dependency relations

Localities can be associated by syntax nesting without being named and referenced. In Figure 1.8 the middle locality is unnamed, unexpressed, unreferenced and hidden inside the syntactic nesting expression. In Figure 1.10 the middle locality is named and explicitly referenced. Locality **C** is dependent on **cross** and **cross** is dependent on **A** and **B**. There are two different syntax expressions in the dependency relations dependently associated by correspondence of the name **cross**.


```
        /*  localities  */
(A/{1 0}/   B/{1 0}/   cross/{11 10 01 00}/
                                C/{1 0}/)
        /*  dependency relations  */
        cross/{11<=[A/1  B/1]
               10<=[A/1  B/0]
               01<=[A/0  B/1]
               00<=[A/0  B/0]  }
        C/{1<=[cross/01  cross/10]
           0<=[cross/00  cross/11] }
```

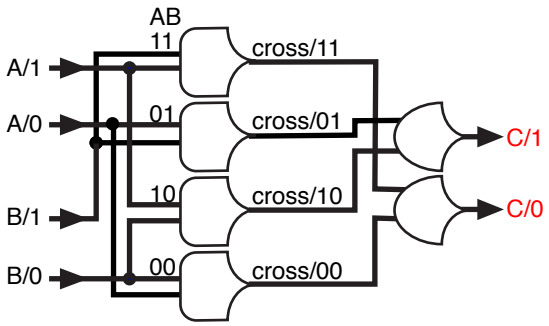


Figure 1.10. Name correspondence.

The expressions of Figure 1.8 and Figure 1.10 illustrates expressional tradeoff between name correspondence expression and syntax expression.

1.4.11. Generality of result locality

Any mapping of cross association appreciation to any result locality differentness can be expressed. For instance C might be a three differentness locality.

```
        /*  localities  */
(A/{1 0}/   B/{1 0}/   C/{2 1 0}/)
        /*  dependency relations  */
        C/{2<=[A/0 B/1] [A/1 B/0]}
           1<=[A/0 B/0]
           0<=[A/1 B/1]  }
```

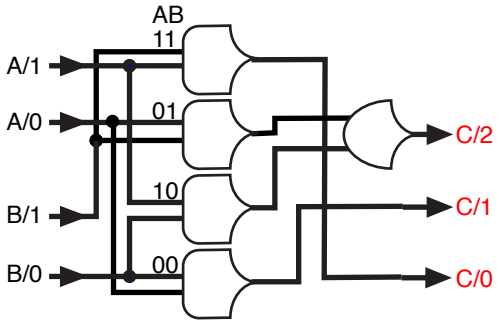


Figure 1.11. Alternative mapping.

The appreciation of the interacting differentnesses can be the asserted recognition of the expression without any further mapping.

```
        /*  localities  */
(A/{1 0}/   B/{1 0}/   cross/{11 10 01 00}/)
        /*  dependency relations  */
        cross/{11<=[A/1 B/1]
               10<=[A/1 B/0]
               01<=[A/0 B/1]
               00<=[A/0 B/0]  }
```

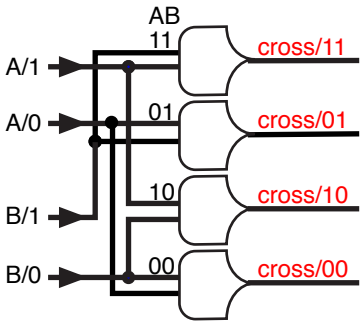


Figure 1.12. No specific mapping.

1.4.12. Selective dependency

Localities can flow as a whole selectively dependent on the differentnesses of one interaction locality cross associating its differentnesses with wholeness references to the other interaction localities.

1.4.12.1. Enable

Only one of three dependencies **A=>D, B=>E or C=>F** is enabled to flow.

```
        /*  localities  */
(A/{1 0}/  B/{1 0}/   C/{1 0}/
 D/{1 0}/   E/{1 0}/ E/{1 0}/
  enable/{D E F}/)
        /*  dependency relations  */
        D<=[A  enable/D]
        E<=[B  enable/E]
        F<=[C  enable/F]
```

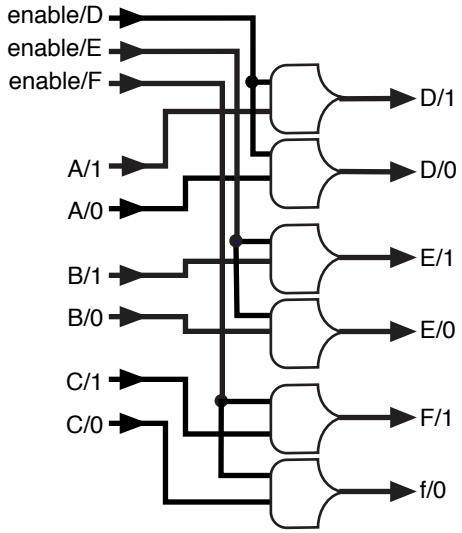


Figure 1.13. Many to many conditional dependency.

1.4.12.2. Steer

Locality **A** is steered to flow to locality **C**, locality **D** or locality **E**.

```
        /*  localities  */
(A/{1 0}/   C/{1 0}/   D/{1 0}/   E/{1 0}/
  steer/{C D E}/)
        /*  dependency relations  */
        C<=[A  steer/C]
        D<=[A  steer/D]
        E<=[A  steer/E]
```

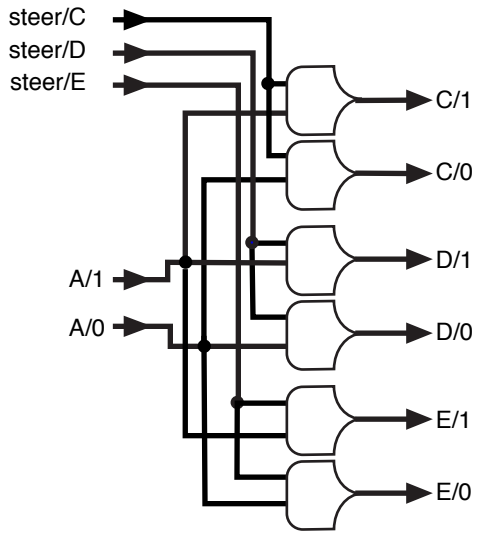


Figure 1.14. One to many conditional dependency.

1.4.12.3. Select

Locality **A**, locality **B** or locality **C** is selected to flow to locality **Z**.

```
        /*  localities  */
( A/{1 0}/   B/{1 0}/   C/{1 0}/   Z/{1 0}/
  select/{A B C}/ )
        /*  dependency relations  */
        Z<={[A  select/A]
            [B  select/B]
            [C  select/C] }
```

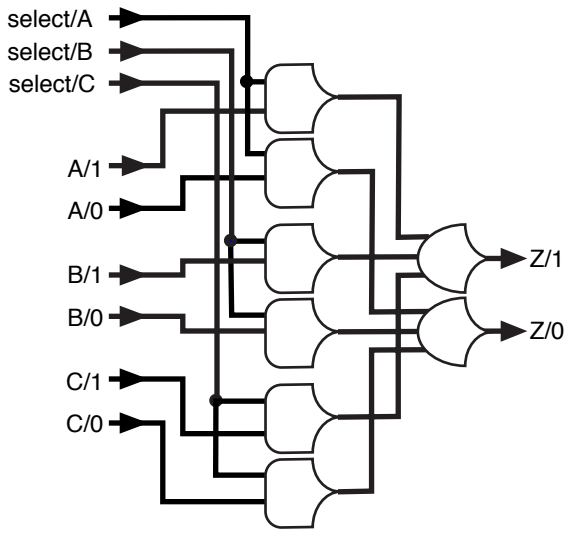


Figure 1.15. Many to one conditional dependency.

1.4.13. General mapping interaction

In Figure 1.16 shows interaction expression and their networks derived from a general mapping table. Each possible combination of cross associated interacting locality differentnesses is recognized and mapped to a specified result differentnesses.

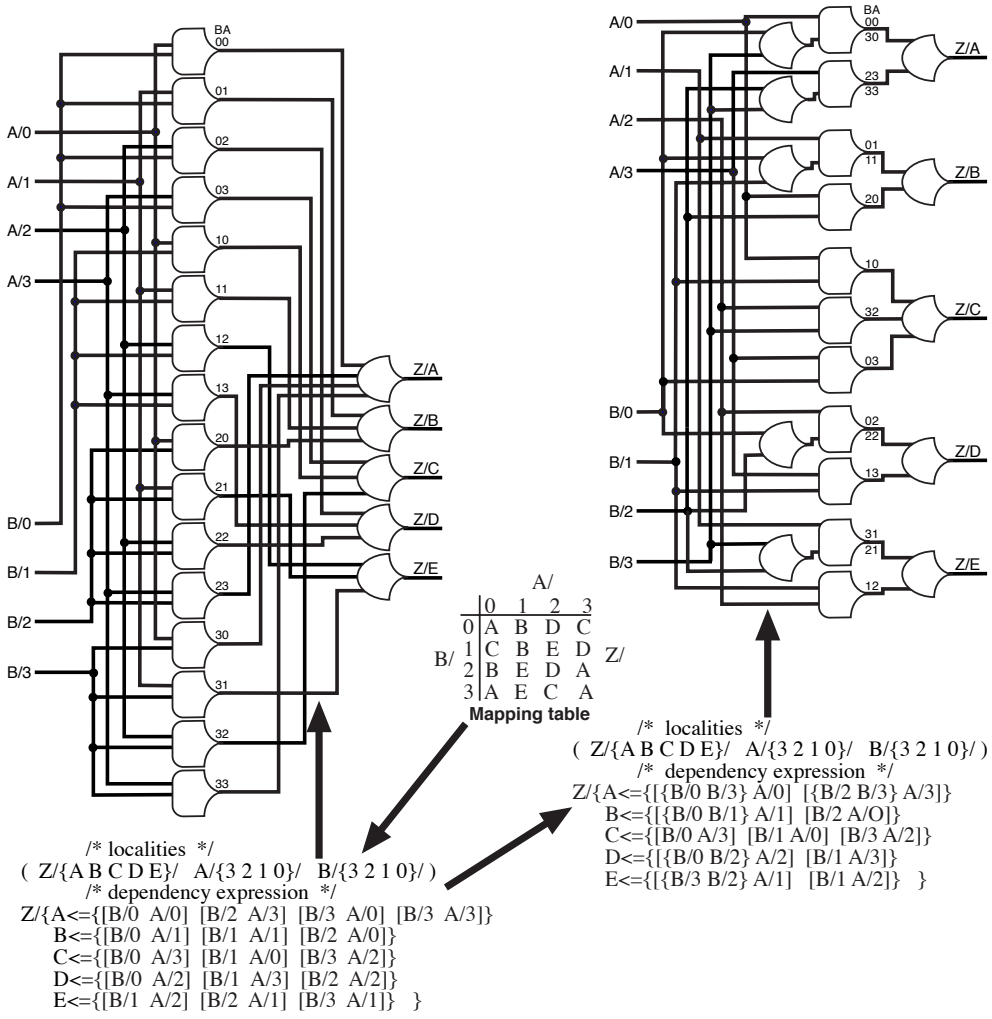


Figure 1.16. Generality and flexibility of expressing interaction mapping.

The expressions, derived from the table in the center, are behaviorally identical. The left expression and network is a direct mapping of the reference table. The right expression and network reflects some consolidation of subexpression.

1.5. The binding portal

In Figure 1.8 **C** is dependent on **A** and **B** but what are **A** and **B** dependent on and what is dependent on **C**. At this point a dependency expression is dependent on an agency extrinsic to itself for the monotonic transitioning of the differentnesses of its interacting localities and for maintaining each transition long enough for the result locality transition to occur. The binding portal exposes the expression to this extrinsic agency allowing its behavior into the expression and also isolating the expression from all other extrinsic influence.

1.5.1. The expression format

Figure 1.17 shows the template for dependency expression which adds a binding portal to the dependency relations introduced above providing an extrinsically referencable expression name

and a syntactic structure of intrinsic localities exposed to the extrinsic reference. To facilitate the text discussion and to make the flow of dependency conveniently visible in the dependency expression interacting localities will be colored blue and result localities will be colored red.

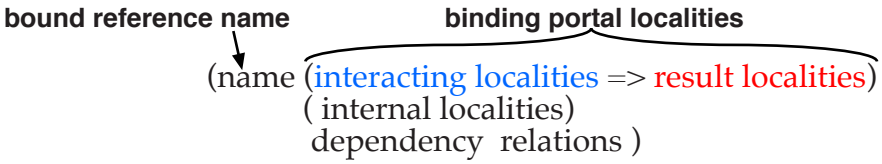


Figure 1.17. Network expression template.

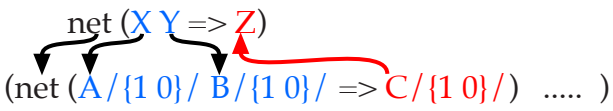
1.5.2. Referencing an expression

The binding portal enables its expression to be referenced extrinsically in terms of name correspondence and syntax structure correspondence. In the examples below the expression is referenced with the corresponding name **net**. The intrinsic locality names **A**, **B** and **C** are associated to the differently named extrinsic localities **X**, **Y** and **Z** by corresponding syntax structure.

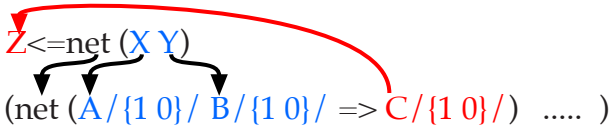
1.5.2.1. One to one reference associations

There are three ways of referencing a network expression through its binding portal:

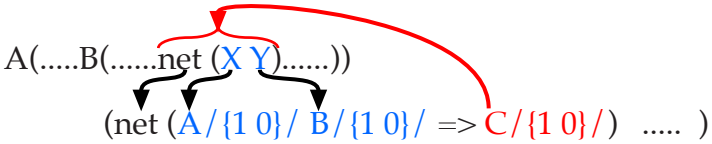
referenced with full portal reference,



referenced with locality nesting,



or referenced with reference nesting.



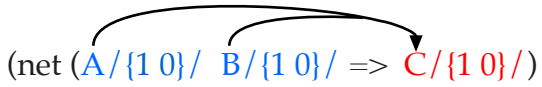
With full portal reference and locality nesting reference the result locality named **C** is explicitly associated to the extrinsic locality named **Z** which can further distribute the differentness of locality **C** with name correspondence. Reference nesting is one to one syntactic association of an unnamed locality which does not support further distribution of the differentness of locality **C**. All of these variations of reference are used in the examples.

The intrinsic localities **A** and **B** are dependent on the extrinsic localities **X** and **Y** which present the monotonic transitioning interaction differentnesses. Intrinsic locality **C** is dependent on intrinsic localities **A** and **B**. Extrinsic locality **Z** is dependent on intrinsic locality **C**.

1.5.2.2. Many to one dependency convergence association

The binding portal expresses a many to one association of interaction localities to a result locality representing the convergence of dependency through the interaction dependency relations.

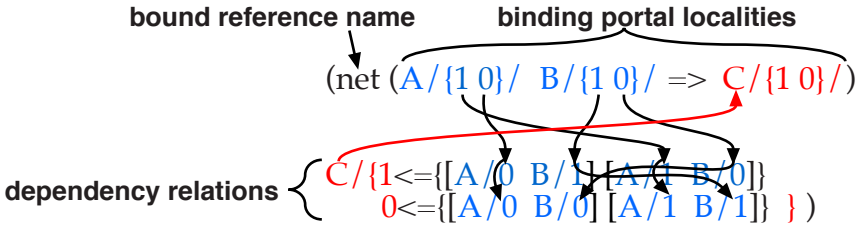
Many to one convergent association



1.5.2.3. One to many dependency divergence association

The binding portal expresses one to many association relations associating each interacting locality by name correspondence to multiple places of association within the expression of dependency relations representing the divergence of dependencies into the interaction dependency relations.

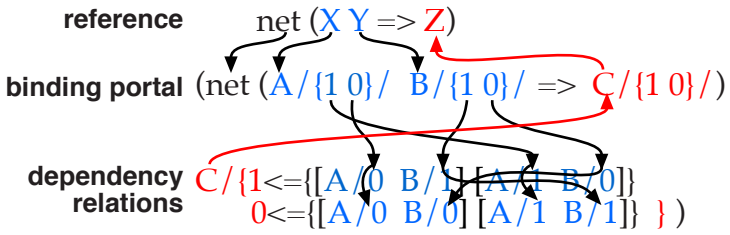
One to many divergent associations



1.5.2.4. The binding portal view of dependency

The binding portal is a collaboration between corresponding names associating different syntax structures and syntax structure associating different names. The extrinsic reference syntax structure is associated to the binding portal syntax structure by name correspondence. The binding portal locality names are associated to the different extrinsic locality names by corresponding syntax structure then the binding portal locality names are associated to the intrinsic dependency relation syntax structures by name correspondence. **Z** is dependent on **C** which is dependent on the dependency relations which are dependent on **A** and **B** which are dependent on **X** and **Y** all of which are dependent on the reference to **net**.

The binding portal and its dependency relations



1.5.2.5. Isolation

The binding portal isolates, bounds and represents its expression as a whole. The bound reference name is the only name in an expression that is extinsically exposed. All other names

within the expression are isolated by the binding portal and can only correspond within the expression. The same names can be reused extinsically without ambiguity. For instance

net(A B => C)

is a valid reference to **net**. The intrinsic **A**, **B** and **C** are syntactically isolated from and different from but also associated to the extrinsic **A**, **B** and **C**.

1.5.2.6. Template expression

The expressions above are recapitulated according to the expression template with the binding portal.

Expression of Figure 1.8

```
(net1(A/{1 0}/  B/{1 0}/ => C/{1 0}/)
        C/{1<={[A/0 B/1] [A/1 B/0]}
          0<={[A/0 B/0] [A/1 B/1]} }  )
```

Expression of Figure 1.10

```
(net2(A/{1 0}/  B/{1 0}/ => C/{1 0}/)
        ( cross/{3 2 1 0}/ )
        cross/{3<=[A/1 B/1]
                2<=[A/1 B/0]
                1<=[A/0 B/1]
                2<=[A/0 B/0]  }
        C/{1<={cross/2  cross/1}
          0<={cross/3  cross/0} }  )
```

Expression of Figure 1.11

```
(net3(A/{1 0}/  B/{1 0}/ => C/{2 1 0}/)
        C/{2<={[A/0 B/1] [A/1 B/0]}
          1<=[A/0 B/0]
          0<= [A/1 B/1]  }  )
```

Expression of Figure 1.12

```
(net3(A/{1 0}/  B/{1 0}/ => cross/{11 10 01 00}/)
        cross/{11<=[A/1 B/1]
                10<=[A/1 B/0]
                01<=[A/0 B/1]
                00<=[A/0 B/0]  } )
```

Expression of Figure 1.13

```
(net4(A/{1 0}/ B/{1 0}/ C/{1 0}/
        enable/{D E F}/  =>
                D/{1 0}/  E/{1 0}/  F/{1 0}/)
        D<=[A  enable/D]
        E<=[B  enable/E]
        F<=[C  enable/F]  )
```

Expression of Figure 1.14

```
(net4(A/{1 0}/  steer/{C D E}/ =>
                C/{1 0}/  D/{1 0}/  E/{1 0}/)
        C<=[A  steer/C]
        D<=[A  steer/D]
        E<=[A  steer/E]  )
```


Expression of Figure 1.15

```
(net5(A/{1 0}/  B/{1 0}/  C/{1 0}/
        select/{A B C}/ => Z/{1 0}/)
        Z<={[A  select/A]
             [A  select/B]
             [A  select/C] } )
```

Expression of Figure 1.16

```
(net6(A/{3 2 1 0}/  B/{3 2 1 0}/ => Z/{A B C D E}/ )
Z/{A<={[{B/0 B/3} A/0]  [{B/2 B/3} A/3]}
   B<={[{B/0 B/1} A/1]  [B2 A/0] }
   C<={[B/0 A/3] [B/1 A/0] [B/3 A/2] }
   D<={[{B/0 B/2} A/2]  [B1 A/3] }
   E<={[{B/3 B/2} A/1]  [B2 A/3] }  )
```

1.5.3. Binding portal composition

The binding portal establishes a new component of compositional reference. At this stage all expressions are constant always providing the same result differentness for the same presented interacting differentnesses. This constancy enables the expression to be indiscriminately referenced from and indiscriminately copied to anywhere and anywhen. In particular, it enables the composition of larger expressions of dependency relations in terms of referencing the binding portals of smaller expressions. References and their dependency relations can be expressed in multiple ways. The four expressions below compose three binding portal references to the Figure 1.8 **net1** expression above to express the network of Figure 1.18.

bignetA expression 1
with full portal reference

```
(bignetA(W X Y Z => C)
        ( A   B )
        net1(W X=> A)
        net1(Y  Z => B)
        net1(A B => C)   )
```

bignetA expression 2
With locality nesting

```
(bignetA(W X Y Z => C)
        ( A   B )
        A<=net1(W X)
        B<=net1(Y  Z)
        C<=net1(A B)   )
```

bignetA expression 3
With reference nesting and locality nesting

```
(bignetA(W X Y Z => C)
        C<=net1(net1(W X) net1(Y Z))   )
```

bignetA expression 4
With reference nesting within full portal reference

```
(bignetA(W X Y Z => C)
        net1(net1(W X)  net1(Y Z)  =>  C)  )
```

All four expressions express the same network.

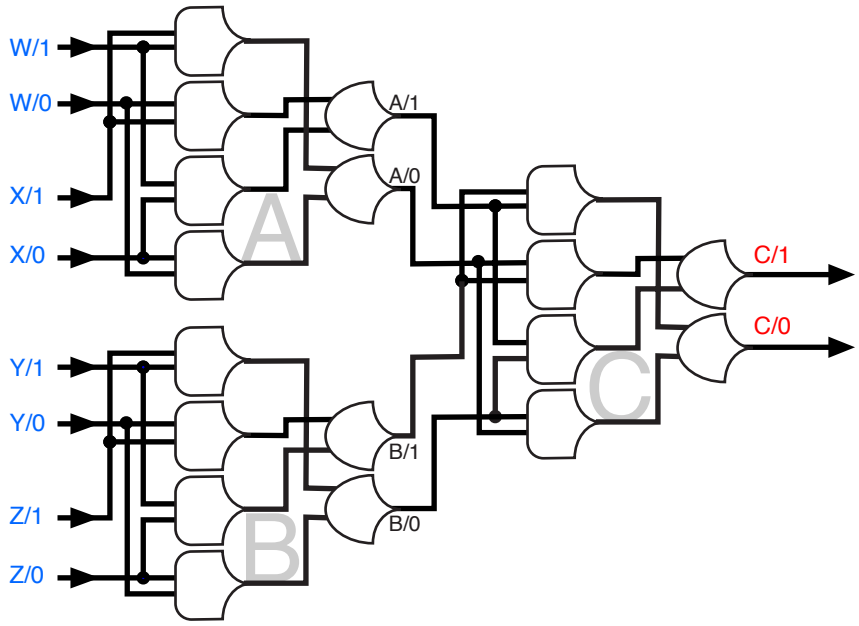


Figure 1.18. BignetA composed from three net1 networks.

1.5.4. Binding portal locality inheritance

In **bignetA** the dependencies are in terms of whole localities and the expressions reference whole localities by name without expressing the structure of the localities but the localities must ultimately be fully expressed to be properly realized. The structures of localities associated through a binding portal must match. When a partially expressed locality is associated to a fully expressed locality through a binding portal the partially expressed locality inherits expression from the fully expressed locality. The above localities **A**, **B** and **C** associate to and inherit their locality expression from the fully expressed locality **net1/C/{1 0}/**. Similarly the above localities **W** and **Y** associate to and inherit from **net1/A/{1 0}/** and localities **X** and **Z** associate to and inherit from **net1/B/{1 0}/**.

Expression 1 locality inheritance expansion:

```
(bignetA(W  X  Y  Z => C)          (bignetA(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/ => C/{1 0}/ )
        ( A   B )                          ( A{1 0}/   B{1 0}/ )
        net1(W X=> A)                      net1(W X=> A)
        net1(Y  Z => B)                    net1(Y  Z => B)
        net1(A B => C)   )                 net1(A B => C)   )
```


Expression 2 locality inheritance expansion:

```
(bignetA(W  X  Y  Z => C)          (bignetA(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/ => C/{1 0}/ )
        ( A   B )                          ( A{1 0}/   B{1 0}/ )
        A<=net1(W X)                       A<=net1(W X)
        B<=net1(Y  Z)                      B<=net1(Y  Z)
        C<=net1(A B)   )                   C<=net1(A B)   )
```

Expression 3 locality inheritance expansion:

```
(bignetA(W  X  Y  Z => C)          (bignetA(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/ => C/{1 0}/ )
   C<=net1(net1(W X) net1(Y Z)) )     C<=net1(net1(W X) net1(Y Z))   )
```

Notice that **A** and **B** are not explicitly expressed but the two ends of a reference nesting must still have matching locality structures.

Expression 4 locality inheritance expansion:

```
(bignetA(W  X  Y  Z => C)          (bignetA(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/ => C/{1 0}/ )
  net1(net1(W X)  net1(Y Z)  =>  C) )     net1(net1(W X)  net1(Y Z)  =>  C) )
```

All of the **bignetA** expressions now have fully expressed binding portals and can pass on inheritance when referenced to compose an even larger network.

1.5.5. Binding portal expression inheritance

The above expression expresses dependency relations among binding portal references each of which has its own expression of dependency which can be substituted for the binding portal reference.

Expressions 1 dependency expression inheritance expansion:

```
(bignetA(W  X  Y  Z => C)          (bignetA2(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/  => C/{1 0}/)
        ( A   B )                          ( A/{1 0}/   B/{1 0}/ )
        net1(W X=> A)                      A/{1<={[W/0  X/1] [W/1  X/0]}
        net1(Y  Z => B)                       0<={[W/0  X/0] [W/1  X/1]} }
        net1(A B => C)   )                 B/{1<={[Y/0  Z/1] [Y/1  Z/0]}
                                              0<={[Y/0  Z/0] [Y/1  Z/1]} }
                                           C/{1<={ [A/0  B/1]  [A/1  B/0] }
                                              0<={ [A/0  B/0]  [A/1  B/1] } } )
```

The expression of a binding portal may include further binding portal references. Binding portal inheritance continues until the binding portal expressions do not include any further binding portal references but are fully primitive expressions.

Inheritance of nested expression is less straightforward and the expression should be converted to minimal syntax form before inheritance expression substitution (see section 1.9.3).

1.5.6. Direct primitivity

bignetA can also be expressed entirely in terms of references to primitive localities.


```
(bignetA1(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/  => C/{1 0}/)
        C/{1<={ [ {[W/0  X/1] [W/1  X/0]}  {[Y/0  Z/0] [Y/1  Z/1]} ]
                [ {[W/0  X/0] [W/1  X/1]}  {[Y/0  Z/1] [Y/1  Z/0]} ] }
           0<={ [ {[W/0  X/1] [W/1  X/0]}  {[Y/0  Z/1] [Y/1  Z/0]} ]
                [ {[W/0  X/0] [W/1  X/1]}  {[Y/0  Z/0] [Y/1  Z/1]} ] } }  )
```

It might be of some interest that this expression is more expensive than the expansion of expression 1 above because of the one to many associations of localities **A** and **B** are now expressed as dependency expressions instead of as locality expressions.

1.5.7. The extrinsic and intrinsic binding portal

The binding portals of the **net1** references are no longer exposed to an extrinsic agency but are intrinsic to a greater network expression with their dependencies fulfilled by the greater network expression itself. However the greater network as a whole still has an exposed binding portal dependent on behavior extrinsic to the network.

The greater expression still assumes that the extrinsic agency will transition interaction localities monotonically between **D** completeness and completely **N** and that each transition will be held long enough for the transition to propagate through the network and transition the result locality to completeness. If this assumption is fulfilled for the greater expression extrinsically exposed binding portal then the identical assumption is fulfilled for all intrinsic binding portal references within the greater expression.

At this stage the extrinsic agency is in complete control of the network. No matter how complex an expression becomes it remains dependent on an uncharacterized extrinsic agency beyond its exposed binding portal for its liveness (no interaction locality presentation - no network behavior), its temporal reference (the monotonic transitions between **D** completeness and completely **N**) and to provide variability of behavior (The behavior of a constant network varies only with the variability of its presented interaction locality differentnesses).

1.5.8. The completeness criterion

At this stage because an expression it completely dependent on the extrinsic agency every expression is constant and fulfills the completeness criterion. The transition to completeness of the result locality implies that the interacting localities have transitioned to completeness, that the expressed interaction is complete and that the result locality has transitioned to the correct result differentness for the presented interacting differentnesses.

Appreciating the transition to completeness of the result first level locality appreciates the completeness of the interaction expression as a whole and is the first step towards expressional autonomy.

1.5.9. Appreciating first level locality transition completeness

Completeness of transition of a locality is appreciated by transformation of the expression of the locality into a dependency expression by forming all of the possible references to locality differentness within the primitive association relations of the locality expression.

Consider locality:

A/{1 0}/

The locality expression is being transformed into references to the locality which do not include the **{D N}** term so the dangling slash is dropped from the locality expression.

A/{1 0}

Then distribute the locality name A/ to form first level locality references within the **"one of"** relation forming a dependency expression that recognizes the transition to **D** completeness and the transition to completely **N** for the locality.

{A/1 A/0}

Express a completeness locality with a single place of association assigned the same name with a **.comp** suffix as

A.comp/

and make it dependent on the completeness expression.

A.comp<={A/1 A/0}

A question mark followed by a locality name, **?localityname**, represents the completeness expression for the referenced locality derived from its locality expression. The completeness of **A** is expressed as:

A.comp<=?A

The resulting completeness expression maps to the locality completeness network of Figure 1.19 labeled **binary**. Locality **A.comp** transitions to **D** when the locality **A** transitions to **D** completeness and transitions to **N** when locality **A** transitions to completely **N**. If locality **A** is a result locality for a dependency network expression then the transition of completeness locality **A.comp** to **D** represents the completeness of the network interaction and the transition of **A.comp** to **N** indicates the readiness of the network for a next interaction.

Derivation of the locality completeness network of Figure 1.19 labled **quaternary**.

```
B/{3-0}/  B.comp/
B/{3-0}
B/{3  2  1  0}
{B/3  B/2  B/1  B/0}
B.comp<={B/3  B/2  B/1  B/}
        /*  expressed as  /
B.comp<=?B
```

Derivation of the locality completeness network of Figure 1.19 labled **decimal**.


```
C/{9-0}/ C.comp/
C/{9-0}
C/{9 8 7 6 5 4 3 2 1 0}
{C/9 C/8 C/7 C/6 C/5 C/4 C/3 C/2 C/1 C/0}
C.comp<={C/9 C/8 C/7 C/6 C/5 C/4 C/3 C/2 C/1 D/0}
        /* expressed as /
C.comp<=?C
```

Derivation of the locality completeness network of Figure 1.19 labled **hexadecimal**.

```
D/{F-A 9-0}/ D.comp/
D/{F-A 9-0}
D/{F E D C B A 9 8 7 6 5 4 3 2 1 0}
{D/F D/E D/D D/C D/B D/A D/9 D/8 D/7 D/6 D/5 D/4 D/3 D/2 D/1 D/0}
D.comp<={D/F D/E D/D D/C D/B D/A D/9 D/8 D/7 D/6 D/5 D/4 D/3 D/2 D/1 D/0}
        /* expressed as /
D.comp<=?D
```

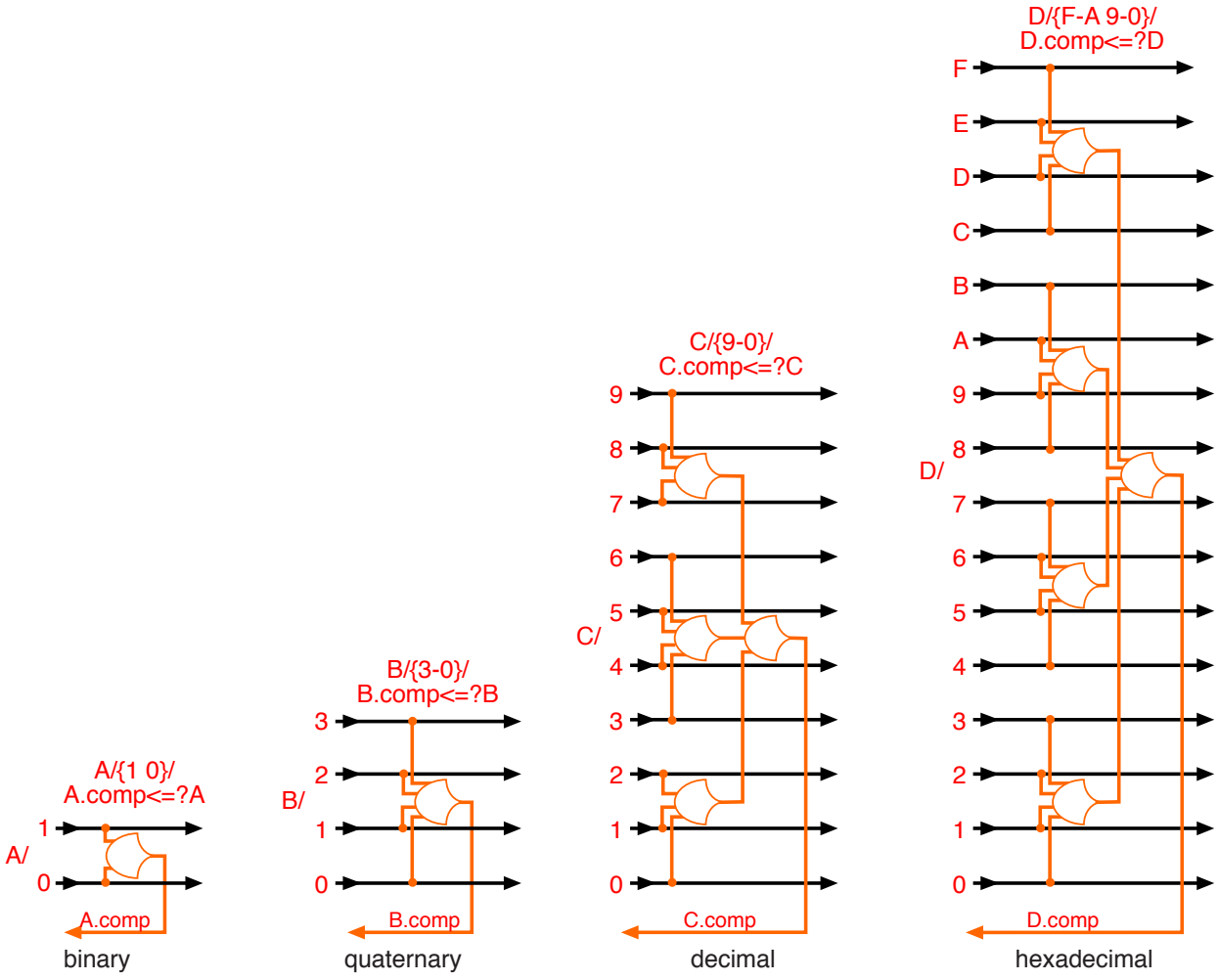



Figure 1.19. Completeness networks derived from their locality expressions.

Each transition of a **.comp** completeness locality is a singular appreciation of the singularly appreciable transitions to **D** completeness and to completely **N** of its referenced locality**.**

1.6. The oscillation network expression: self regulation

The appreciation of the completeness transitioning of its result locality empowers an expression fulfilling the completeness criterion to self regulate the acceptance of its interaction locality transitions with its result completeness locality feeding back to and closing with its interaction localities through **"all of"** behaviors. After the result locality transitions to **D** completeness the interaction localities can be enabled to transition to completely **N**. After the result locality transitions to completely **N** the interaction localities can be enabled to transition to **D** completeness. This enabling involves converting the completeness locality condition from **D** to **N** and from **N** to **D**. This conversion that initializes to **N** is represented with the syntax symbol ~.

The closed expression with a single conversion is similar to a spontaneously transitioning binary ring oscillator circuit with a single inversion so is called an *oscillation network* spontaneously alive continually striving to accept transitioning interaction differentnesses.

The term **C.comp**<=?**C** expresses the completeness reduction of the locality **C** to the completeness locality **C.comp**. The term **[Ain ~C.comp]** expresses enabling the next transition of interaction locality **Ain** through an **"all of"** behavior.

The oscillation network dependency expression:

```
(net(Ain/{1 0}/ Bin/{1 0}/=> C/{1 0}/)
  (A/{1 0}/  B/{1 0}/  C.comp/ )
        A<=[Ain ~C.comp]
        B<=[Bin ~C.comp]
        C.comp<=?C/{1<={{[A/0  B/1] [A/1  B/0]}
                   0<={[A/0  B/0] [A/1  B/1]} } )
```

expresses the network:

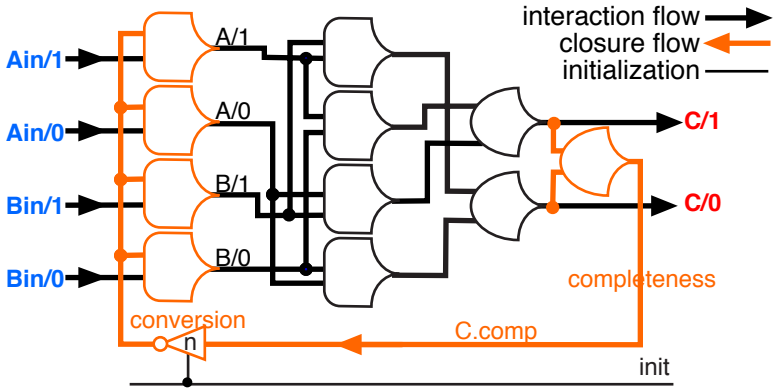


Figure 1.20. Self regulating oscillation network.

The **black** paths are interaction transition flow paths. The **orange** paths are closure transition paths which flow in the opposite direction.

C.comp is dependent on the completeness of **C** the completeness of which is dependent on **A** and **B** which are dependent on **Ain** and **Bin** and the conversion of **~C.comp** closing the oscillation network**.**

1.6.1. Oscillation network behavior

At initialization the conversion result is forced to **N**. The initial presentation to **A** and **B** must be completely **N** which is enabled through the **"all of"** behaviors and flows through the network emptying the network of interaction differentness. The result completeness **C.comp** transitions to **N**. When init is released the conversion result of **~C.comp** presented to the **"all of"** behaviors transitions to **D**.

When the interacting localities transition to **D** completeness the transition flows through the rank of **"all of"** behaviors. **~C.comp** will remain **D** until the transition has propagated through the expression and the result locality has transitioned to **D** completeness and **C.comp** has

transitioned to **D** upon which ~**C.comp** transitions to **N**. If **Ain** and **Bin** transition to completely **N** beforehand they will not pass the rank of **"all of"** behaviors until ~**C.comp** transitions to **N**.

When the interacting localities transition to completely **N** the transition flows through the rank of **"all of"** behaviors. ~**C.comp** will remain **N** until the transition has propagated through the expression and the result locality has transitioned to **N** completeness and **C.comp** has transitioned to **N** upon which ~**C.comp** transitions to **D**. If **Ain** and **Bin** transition to completely **D** beforehand they will not pass the rank of **"all of"** behaviors until ~**C.comp** transitions to **D**.

When the interacting localities transition to **D** completeness the transition flows through the rank of **"all of"** behaviors. ~**C.comp** will remain **D** until the transition has propagated through the expression and the result locality has transitioned to **D** completeness and **C.comp** has transitioned to **D** upon which ~**C.comp** transitions to **N**. If **Ain** and **Bin** transition to completely **N** beforehand they will not pass the rank of **"all of"** behaviors until ~**C.comp** transitions to **N**.

When the interacting localities transition to completely **N** the transition flows through the rank of **"all of"** behaviors. ~**C.comp** will remain **N** until the transition has propagated through the expression and the result locality has transitioned to **N** completeness and **C.comp** has transitioned to **N** upon which ~**C.comp** transitions to **D**. If **Ain** and **Bin** transition to completely **D** beforehand they will not pass the rank of **"all of"** behaviors until ~**C.comp** transitions to **D**.

… and so on.

1.6.2. Appreciating intrinsic interaction time

The closure regulates the transition of wavefronts into the network. The oscillation network serves as an *escapement mechanism* establishing a nonuniform metric of interaction time for its encompassed constant expression. Even though the oscillation network is now regulating its own interaction locality transitions and determining its own intrinsic interaction time it is still dependent through its exposed binding portal on an extrinsic agency to transition its interaction localities between **D** completeness and completely **N**.

At this point the closure is intrinsic to the expression. The extrinsic agency does not have access to the closure so must still time itself in relation to its own time metric and with understanding of the network delays. The extrinsic transitions cannot get ahead of the intrinsic closure enables. The extrinsic agency does not have access to the intrinsic closure behavior and there is no explicit coordination between the oscillation expression and the extrinsic agency. It may seem that the emergent oscillation network has not accomplished anything useful, but stay tuned.

1.7. The pipeline network expression: Composing self regulation

Placing the result locality completeness determination of oscillation network **A** after the interaction locality enable closure of oscillation network **B** links the two oscillation networks coordinating interaction transition flow from oscillation network **A** to oscillation network **B**.

Figure 1.21 shows **pink**, **orange** and **blue** oscillation networks which are linked by placing the completeness of the **pink** network after the enable of the **orange** network and by placing the completeness of the **orange** network after the enable of the **blue** network forming a pipeline of linked oscillation networks.

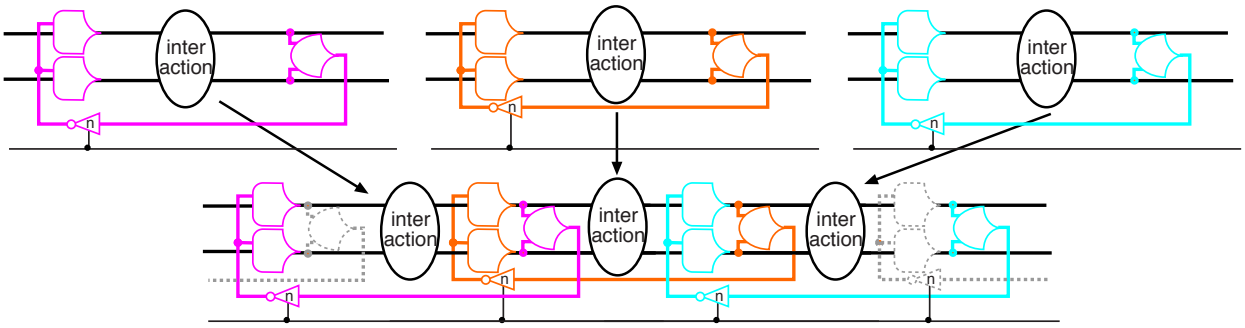


Figure 1.21. linking oscillation networks.

1.7.1. The closure link

Placing the completeness appreciation of the upstream network after the enable of the downstream network forms a closure link the expression of which is derived from the expression of the locality associating the two networks. Three first level locality links with their locality expression are shown in Figure 1.22

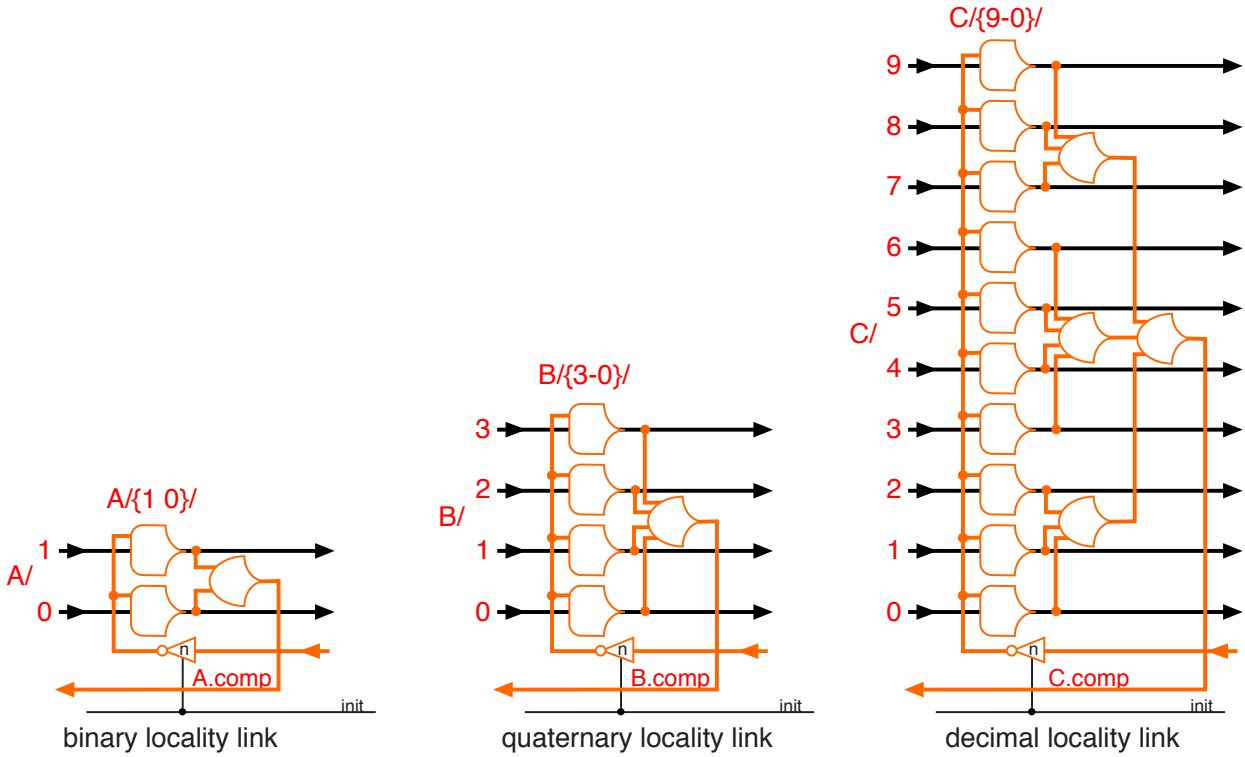


Figure 1.22. First level oscillation network closure links.

The enable expression is all of the possible differentnesses of the locality cross associated with the converted closure from the downstream network with each cross association combination presented to an **"all of"** behavior. Deriving the locality completeness expression from the locality expression was discussed above in section 1.5.9.

1.7.2. Pipeline behavior: wavefronts and bubbles

Each oscillation network of a pipeline network is individually and spontaneously alive continually striving to transition between **D** completeness and completely **N**. As oscillation networks individually oscillate, alternating wavefronts of transition to **D** completeness and completely **N** spontaneously flow from oscillation network to oscillation network through the links and the interaction dependency networks while alternating closure transitions between **D** and **N** called bubbles flow in the opposite direction through links and around the interaction dependency networks.

1.7.2.1. Wavefront flow

Interaction transition wavefronts flow through the pipeline interaction network from oscillation network to oscillation network. The transition wavefront accepted by the **pink** network from a previous network flows to the enable of the **pink** network and is stably maintained until accepted by the **orange** network however long this might take. When accepted by the **orange** network the **pink** network can determine the completeness of its flow and can accept the next wavefront transition into its network.

The transition wavefront accepted by the **orange** network from the **pink** network flows to the enable of the **blue** network and is stably maintained by the **orange** network until accepted by the **blue** network however long this might take. When accepted by the **blue** network the **orange** network can determine the completeness of its flow and can accept the next wavefront transition into its network from the **pink** network.

The transition wavefront accepted by the **blue** network from the **orange** network flows to the enable of a next network and is stably maintained by the **blue** network until accepted by the next network however long this might take. When accepted by the next network the **blue** network can determine the completeness of its flow and can accept the next wavefront transition from the **orange** network which can then accept the next wavefront from the pink network … and so on.

In the meantime the **pink** oscillation network has received its next transition wavefront.

1.7.2.2. Bubble flow

Bubbles are the closure acceptances and enables transitioning between **D** and **N** flowing from link to link around the interaction networks counter to the flow of interaction transition wavefronts.

1.7.3. The counter flowing networks

The interaction wavefront network and the closure bubble network form two counter flowing networks intersecting and mutually regulating each other through the links where the the bubbles enable the wavefront transition to flow and the enabled wavefronts engender the bubble transition flow. In Figure 1.23 the interaction wavefront flow network, shown as **black**, the closure bubble flow network, shown as **orange**, intersect through the **"all of"** enable ranks shown as grey.[5]

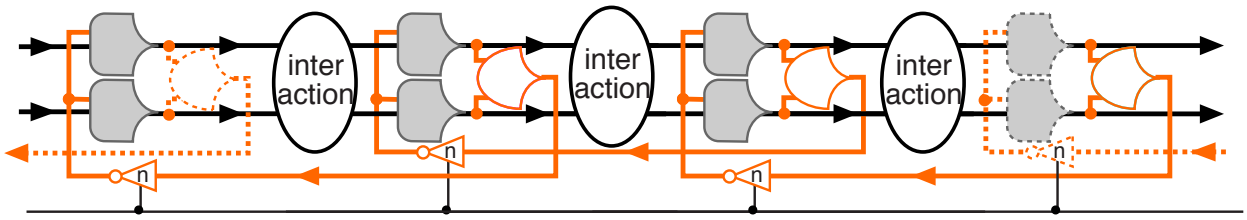


Figure 1.23. The counter flowing networks.

1.7.4. The link expression problem

The problem with the picture in Figure 1.24 showing linked oscillation networks is that the oscillation network expressions reach inside each other to expression the closure links and there is ambiguity about how the links are expressed. The oscillation network expression does not present a coherent unit of composition with a straightforward composition boundary.

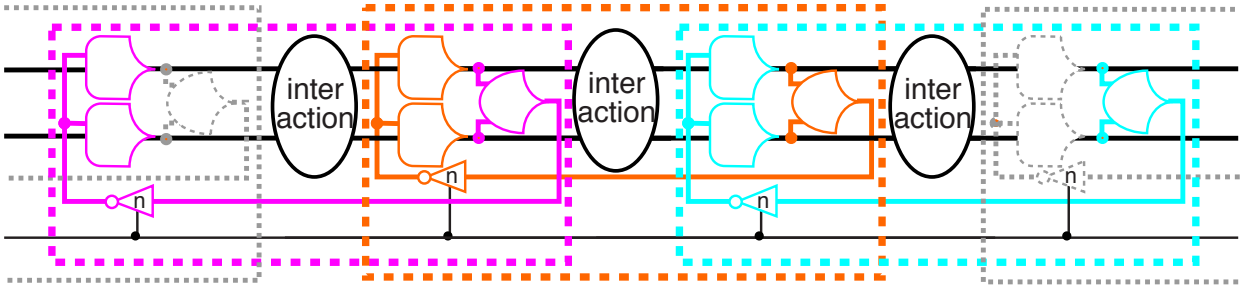


Figure 1.24. Linked oscillation networks.

1.7.5. Half oscillations

If you squint properly you can see the picture somewhat differently in Figure 1.25 as a composition of half oscillations with distinct boundaries that do not overlap presenting a coherent unit of composition.

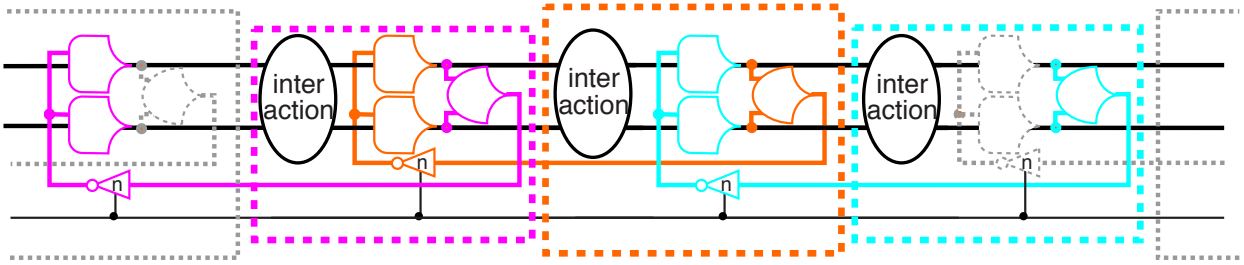


Figure 1.25. Composed half oscillation networks.

A half oscillation expression fully expresses its result locality link but does not express its interaction locality links which will occur as result locality links in other half oscillation expressions. Figure 1.26 illustrates the composition of half oscillations bounded by black dashed lines forming a full oscillation network bounded by a pink dotted line. The interaction locality and the result locality link for the oscillation network are in different half oscillation networks

This means that closure is no longer intrinsic to an expression and that closure transitions as well as interacting wavefront transitions must flow through the binding portal of the half oscillation. In particular, for every interaction wavefront flowing through a binding portal there is an associated closure bubble flowing in the opposite direction.

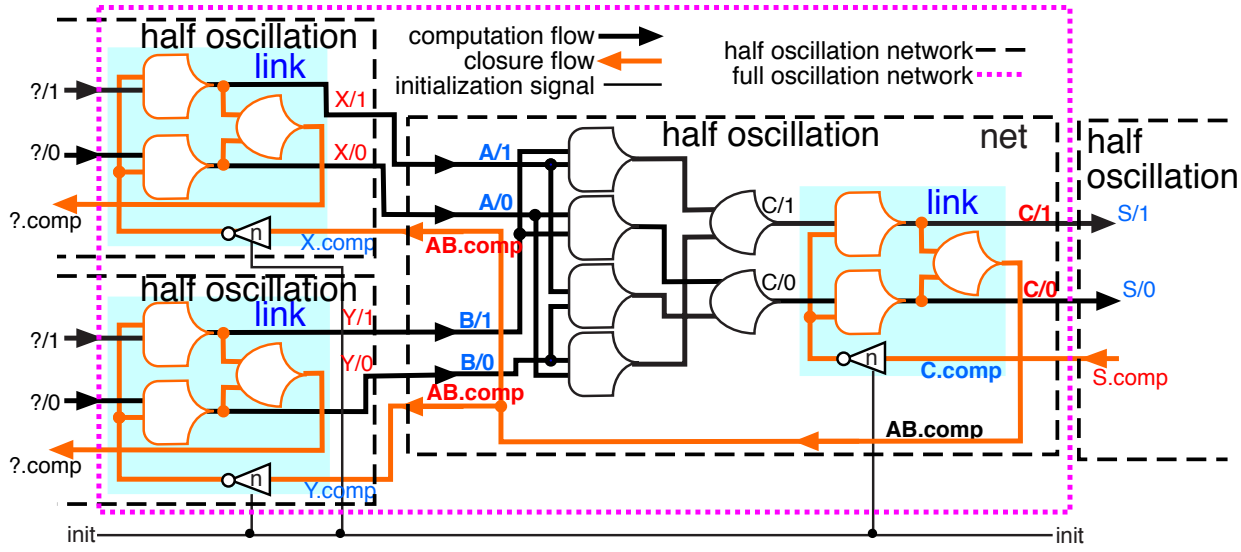


Figure 1.26. Linked half oscillation networks.

There is one complete oscillation network in Figure 1.26 but the expression of the oscillation network occurs over three component half oscillation expressions none of which contains a complete oscillation network. Each component constant network with a *link* on its output locality forms a *pipeline component network* expressing two *half oscillations*, an input/completeness half oscillation closing with the inputs to the network and an output/enable half oscillation closed by the output of the network.

1.7.6. The half oscillation network expression

A half oscillation expresses the interaction dependencies and the enable and completeness of its result locality link as well as the flows through its binding portal. Each locality in the binding portal is now associated with a closure locality by a comma. **A/{1 0}/,AB.comp/** expresses the interaction locality **A/{1 0}/** associated with its completeness closure **AB.comp/**. The colors still represent the direction of transition flow through the binding portal, blue is input and red is output. **C/{1 0}/,C.comp/** expresses the result locality **C/{1 0}/** associated with its completeness closure **C.comp/**. Again **blue** is input and **red** is output.

The interaction dependency relations are inside an **"all of"** behavior with **~C.comp** which is the last dependency applied before the completeness of result locality **C** is appreciated.

```
(net(A/{1 0}/,AB.comp/  B/{1 0}/,AB.comp/ => C/{1 0}/,C.comp/)
        AB.comp<=?[C/{1<={{[A/0  B/1] [A/1  B/0]}
                        0<={[A/0  B/0] [A/1  B/1]} }   ~C.comp] )
```


AB.close is dependent on the completeness reduction **?** of **C** the completeness of which is dependent on the completenesses of **A** and **B** and the conversion ~ of **C.comp.**

The interaction and closure flows through a portal are dependent but are not simultaneous. For instance, in the binding portal **A** is paired with **AB.comp**. In Figure 1.26 an **AB.comp** transition will flow out of the portal and later enable the transition of **X** which will then flow into the portal as a transition of **A**.

The above expresses the half oscillation network **net** of Figure 1.26.

1.7.6.1. The extrinsic agency responsibility: no extrinsic time metric

The extrinsic agency now has access to the closure through the binding portal and the responsibility of the extrinsic agency becomes to hold each interaction locality transition until the completeness of transition of the result locality is indicated by its closure. The extrinsic agency no longer need understand the delay behavior of the expression in terms of its own time metric but must now honor the closures of the expression which is establishing its own understanding of its delay behavior in its own time metric with its own tick of time.

1.7.6.2. The closure protocol

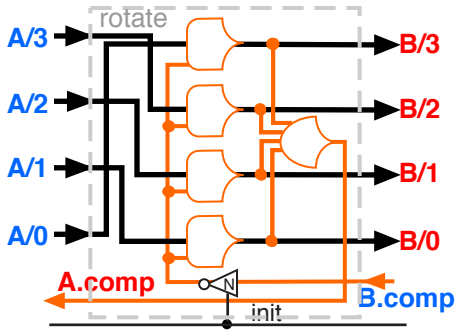
Each locality is dependent on other localities and has other localities dependent on it.

- A locality completeness closes with (regulates monotonic transition of) all localities on which its completeness is dependent.
- A locality is closed by (its monotonic transition regulated) all the localities which depend on it for their own completeness.

1.7.7. Composing half oscillations: a simple pipeline

The half oscillation becomes the new unit of composition. Composing pipelines from half oscillations is illustrated with a simple **rotate** half oscillation that expresses a single step of locality differentness rotation.

```
(rotate(A/{3 2 1 0}/,A.comp/ => B/{3 2 1 0}/,B.comp/)
        A.comp<=?[B/{3<=A/0
                     2<=A/3
                     1<=A/2
                     0<=A/1 }   ~B.comp] )
```



Half oscillation expressions compose by associating a binding portal result locality to a binding portal interaction locality. But now each binding portal locality is paired with a closure locality. Associating half oscillations result locality **rotate/B/,B.comp/** to half oscillation interaction locality **rotate/A/,A.comp/** expresses the pipeline network **rotate2** containing one oscillation network bounded by half oscillations ready for further composition.

Expression 1		Expression 2		Expression 3
(rotate2(X => Y) **Y<=rotate(rotate(X)))**	or	**(rotate2(X => Y)** **rotate(rotate(X) => Y))**	or	**(rotate2(X => Y)** **(C)** **rotate(X => C))** **rotate(C => Y))**

The expressions above reference locality names expressing the dependency relations among whole localities but do not express the structure of the localities or their closure relations. This convenient expression is valid and sufficient because it expresses the essential dependencies. Each locality's structure as well as its closure relations are inherited from fully expressed localities associated through the binding portal references. In this case from reference association to **rotate** whose binding portal is fully expressed.

1.7.7.1. Binding portal inheritance

Locality **rotate2/X** associates to and inherits from **rotate/A/{3 2 1 0}/,A.comp/**. Locality **rotate2/Y** associates to and inherits from **rotate/B/{3 2 1 0}/,B.comp/.** The **rotate2** binding portal for all three forms of the expression above becomes fully formed and ready to further compose.

```
        /* expression 1 */
(rotate2( X/{3 2 1 0}/,X.comp/ => Y/{3 2 1 0}/,Y.comp/)
  Y,Y.comp<=rotate(rotate(X,X.comp))   )

        * expression 2 */
(rotate2( X/{3 2 1 0}/,X.comp/ => Y/{3 2 1 0}/,Y.comp/)
  rotate(rotate(X,X.comp) => Y,Y.comp) )

        * expression 3 */
(rotate2( X/{3 2 1 0}/,X.comp/ => Y/{3 2 1 0}/,Y.comp/)
  ( C/{3 2 1 0}/,C.comp/ )
 rotate(X,X.comp => C,C.comp) )
 rotate(C,C.comp => Y,Y.comp) )
```

All three expressions express the network of Figure 1.27.

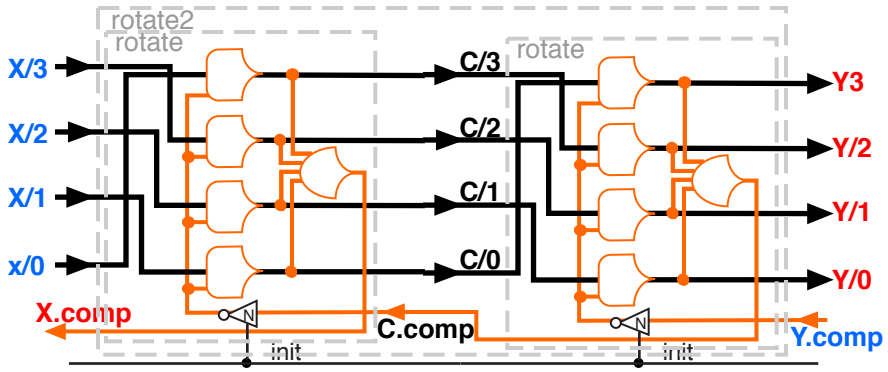


Figure 1.27. The pipeline network expressed by rotate2.

Notice that the composition of half oscillations simultaneously expresses the composition of both the interaction/wavefront network (black) and the closure/bubble network (orange).

1.7.7.2. Composing bigger pipelines

The pipeline **rotate2** is bounded by half oscillations ready for further composition. Each interaction locality and its associated closure is a half oscillation network boundary projecting through the binding portal seeking further composition. Connecting two half oscillations **rotate2/Y/,Y.comp/** to **rotate2/X/,X.comp/** forms the larger pipeline network **rotate4** expressing a four step rotate containing three oscillation networks bounded by half oscillations ready for further composition.

```
(rotate4( S => T)
  T<=rotate2(rotate2(S))  )
```

Another way to express **rotate4** is

```
(rotate4( S => T)
  T<=rotate(rotate(rotate(rotate(S))))  )
```

After inheritance its binding portal is ready for further composition.

```
(rotate4( S/{3 2 1 0}/,S.comp/ => T/{3 2 1 0}/,T.comp/)
  T,T.comp<=rotate2(rotate2(S,S.comp))  )
```

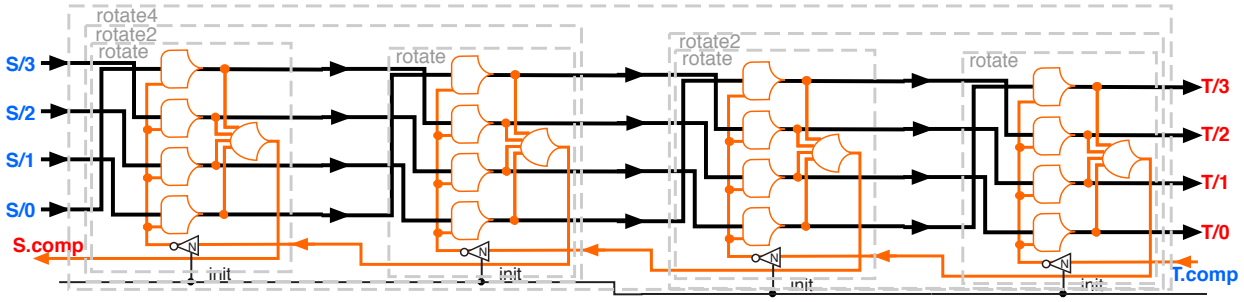


Figure 1.28. The pipeline network expressed by rotate4.

rotate4 can be further composed to **rotate8** and so on.

```
(rotate8( A => B)
  B<=rotate4(rotate4(A))   )
```

Locality **rotate8**/**A** inherits from **rotate4**/**S**/{3 2 1 0}/,**S.comp**/. Locality **rotate8**/**B** inherits from **rotate4**/**T**/{3 2 1 0}/,**T.comp**/.

The inheritance:

```
(rotate8( A/{3 2 1 0}/,A.comp/ => B/{3 2 1 0}/,B.comp/)
  B,B.comp<=rotate4(rotate4(A,A.comp))   )
```

1.7.8. Expressing more complex pipelines

The examples in Figure 1.29 illustrates the possible granularities of intersection between the counter flowing networks for the network of Figure 1.18.

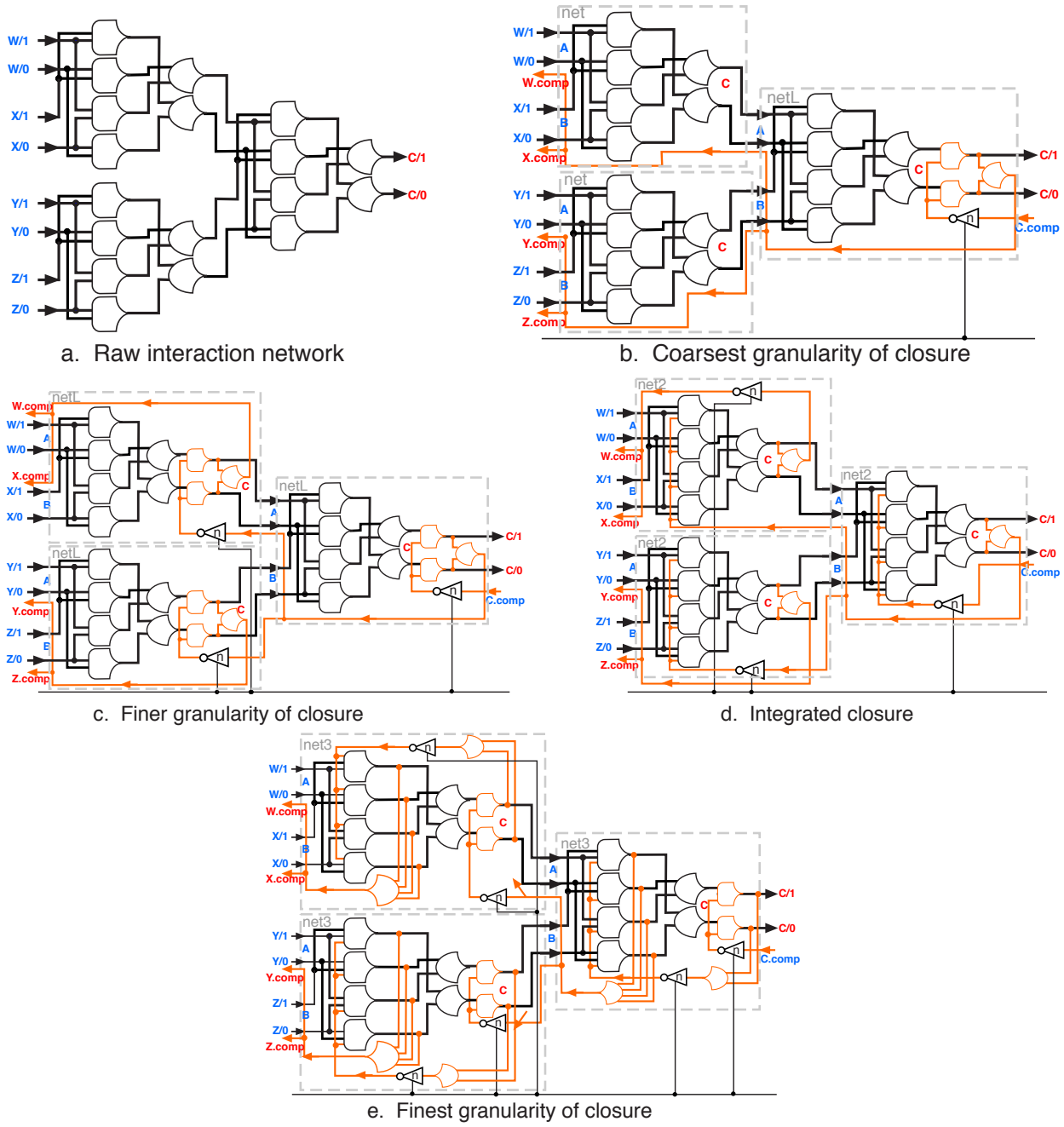


Figure 1.29. The varieties of closure structure for a given network.

1.7.8.1. Network expression Figure 1.29a: no closure

The network is expressed entirely in terms of dependency relations among primitive components with the localities explicitly expressed (see section 1.5.6). No binding portal references and no inheritance.


```
(bigneta(W/{1 0}/  X/{1 0}/  Y/{1 0}/  Z/{1 0}/  => C/{1 0}/)
        C/{1<={ [ {[W/0  X/1] [W/1  X/0]}  {[Y/0  Z/0] [Y/1  Z/1]} ]
                [ {[W/0  X/0] [W/1  X/1]}  {[Y/0  Z/1] [Y/1  Z/0]} ] }
           0<={ [ {[W/0  X/1] [W/1  X/0]}  {[Y/0  Z/1] [Y/1  Z/0]} ]
                [ {[W/0  X/0] [W/1  X/1]}  {[Y/0  Z/0] [Y/1  Z/1]} ] } }  )
```

The subnetwork **net** can be expressed (see section 1.5.3).

```
(net(A/{1 0}/  B/{1 0}/ => C/{1 0}/)
        C/{1<={{[A/0  B/1] [A/1  B/0]}
           0<={[A/0  B/0] [A/1  B/1]} }  )
```

With **bigneta2** dependencies are expressed in terms of references to **net**.

```
(bigneta2(W X Y Z => C)
        C<=net(net(W X) net(Y Z))  )
```

And then inheritance expanded.

```
(bigneta2(W/{1 0}/ X/{1 0}/ Y/{1 0}/ Z/{1 0}/ => C/{1 0}/)
        C<=net(net(W X) net(Y Z))  )
```

bigneta is a dependency network with no closure.

1.7.8.2. Network expression Figure 1.29b: coarsest granularity of closure

Three approaches are presented to expressing the network of Figure 1.29b.

1. The result link can be incorporated into the expression of dependency relations among primitive behaviors. There are no references to already expressed networks and hence no inheritance so this primitive expression of **bignetb** has to explicitly express the closure in the binding portal.

```
 (bignetb(W/{1 0}/,WXYZ.comp/  X/{1 0}/,WXYZ.comp/
          Y/{1 0}/,WXYZ.comp/  Z/{1 0}/,WXYZ.comp/  => C/{1 0}/,C.comp/)
WXYZ.comp<=?[C/{1<={ [ {[W/0  X/1] [W/1  X/0]}  {[Y/0  Z/0] [Y/1  Z/1]} ]
                     [ {[W/0  X/0] [W/1  X/1]}  {[Y/0  Z/1] [Y/1  Z/0]} ] }
                0<={ [ {[W/0  X/1] [W/1  X/0]}  {[Y/0  Z/1] [Y/1  Z/0]} ]
                     [ {[W/0  X/0] [W/1  X/1]}  {[Y/0  Z/0] [Y/1  Z/1]} ] } ~C.comp ] )
```

2. the dependency relations can also be expressed with references to subexpression **net** above which did not express closure relations so expression **bignetb** is the first hierarchical level of closure expression and must explicitly express the closure relations. The color coding represents input and output through binding portals in relations to the expression, in this case **bignetb**.


```
(bignetb(W/{1 0}/,WXYZ.comp/  X/{1 0}/,WXYZ.comp/
          Y/{1 0}/,WXYZ.comp/  Z/{1 0}/,WXYZ.comp/  => C/{1 0}/,C.comp/)
WXYZ.comp<=?[C<=net(net(W  X) net(Y  Z))  ~C.comp] )
```

3. The link can be expressed in a component expression and the component expression incorporated by reference. The expression **netL** incorporates a closure link to the above expression **net**.

```
(netL(A/{1 0}/,AB.comp/  B/{1 0}/,AB.comp/ => C/{1 0}/,C.comp/)
       AB.comp,<=?[C/{1<={{[A/0  B/1] [A/1  B/0]}
                          0<={[A/0  B/0] [A/1  B/1]} }   ~C.comp] )
```

bignetb references **netL** with a link once and **net** without a link twice.

```
(bignetb(W  X  Y  Z  => C)
            C<=netL(net(W X)  net(Y Z)) )
```

bignetb/C inherits from the **netL/C**.

```
(bignetb(W  X  Y  Z  => C/{1 0}/,C.comp/)
            C,C.comp<=netL(net(W X)  net(Y Z)) )
```

The **net/(W X) reference** inherits structure and closure for its result locality **net/C** from **netL/A**. The **net/(Y Z) reference** inherits structure and closure for its result locality **net/C** from **netL/B**. For both inheritances:

```
(net(A/{1 0}/  B/{1 0}/ => C/{1 0}/,C.comp)
       C/{1<={[A/0 B/1] [A/1 B/0]}
          0<={[A/0 B/0] [A/1 B/1]} }  )
```

The inherited closure does not flow through the dependency relations but passes through the binding portal around the dependency relations to close for the interaction localities **net/A** and **net/B**. For both inheritances:

```
(net(A/{1 0}/,C.comp  B/{1 0}/,C.comp => C/{1 0}/,C.comp)
       C/{1<={[A/0 B/1] [A/1 B/0]}
          0<={[A/0 B/0] [A/1 B/1]} }  )
```

Expression **net** now has an inheritable binding portal and **bignetb** interaction localities can inherit from the expanded **net** expression.

bignetb/W and **bignetb/X** inherits from the the reference to **net/(W X)** .
bignetb/Y and **bignetb/Z** inherits from the the reference to **net/(Y Z)** .

The inheritance expanded network expression.


```
(bignetb(W/{1 0}/,W.comp/  X/{1 0}/,X.comp/
                    Y/{1 0}/,Y.comp/  Z/{1 0}/,Z.comp/  => C/{1 0}/,C.comp/)
      C,C.comp<=netL(net(W,W.comp  X,X.comp) net(Y,Y.comp  Z,Z.comp)) )
```

Now **bignetb** has a fully formed binding portal and is ready for further composition.

Once closure has appeared in a binding portal it persists and propagates by inheritance. It cannot be unexpressed.

bignetb is a half oscillation network.

1.7.8.3. Network expression Figure 1.29c: finer granularity of closure

bignetc of Figure 1.29c is composed with three references to **netL** above. The expression of **bignetc** inherits its locality structure and closure relations from the references to **netL**.

```
(bignetc(W  X  Y  Z  => C)
            C<=netL(netL(W  X) netL(Y  Z))  )
```

bignetc/C inherits from **netL/C**
Bignetc/W and **bignetc/Y** inherit from **netL/A**
Bignetc/X and **bignetc/Z** inherit from **netL/B**

```
(bignetc(W/{1 0}/,W.comp/  X/{1 0}/,X.comp/
                Y/{1 0}/,Y.comp/  Z/{1 0}/,Z.comp/  => C/{1 0}/,C.comp/)
      C,C.comp<=netL(netL(W,W.comp  X,X.comp) netL(Y,Y.comp  Z,Z.comp)) )
```

bignetc is a composition of three half oscillation networks that contains one full oscillation network bounded by half oscillations ready for further composition with other half oscillations.

1.7.8.4. Network expression Figure 1.29d: integrating the link

The **"all of"** rank of a cross association search can serve as the enable behavior of a link. This can save some primitive behaviors with the tradeoff of increasing the inputs of other primitive behaviors. Whether such a tradeoff is useful depends on the specifics of a network. **bignetd** is composed from component network **net2** that contains the alternatively structured links.

In the expression of **net2** below ~C.close is applied to the cross association rank of **"all of"** behaviors that are determining the differentness of locality C. The rank of **"all of"** behaviors does double duty as interaction behaviors and as closure enable behaviors.

```
(net2(A/{1 0}/,AB.comp/  B/{1 0}/,AB.comp/ => C/{1 0}/,C.comp/)
      (  allofrank/{3 2 1 0}/  )
      [allofrank/{3<=[A/1  B/1]
                  2<=[A/0  B/1]
                  1<=[A/1  B/0]
                  0<=[A/0  B/0]  }  ~C.comp]
      AB.comp<=?C/{1<={allofrank/1  allofrank/2}
                   0<={allofrank/0  allofrank/3} } )
```


Expressed network

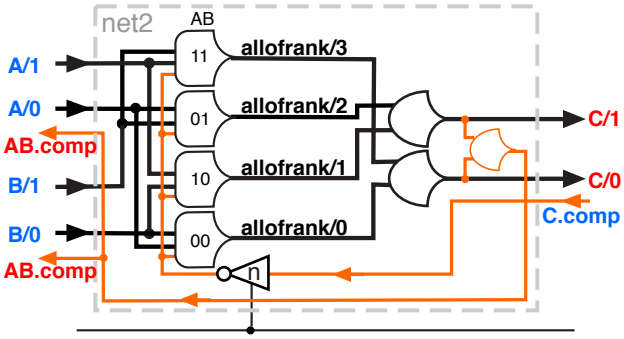


Figure 1.30. net2 network.

bignetd is composed from three references to **net2**.

```
(bignetd(W  X  Y  Z  => C)
              C<=net2(net2(W X) net2(Y Z))  )
```

bignetd/C inherits from **net2/C**
bignetd/W and **bignetd/Y** inherit from **net2/A**
bignetd/X and **bignetd/Z** inherit from **net2/B**
The inheritance expanded network expression.

```
 (bignetd(W/{1 0}/,W.comp/  X/{1 0}/,X.comp/
                Y/{1 0}/,Y.comp/  Z/{1 0}/,Z.comp/  => C/{1 0}/,C.comp/)
        C,C.comp<=net2(net2(W,W.comp  X,X.comp) net2(Y,Y.comp  Z,Z.comp)) )
```

bignetd contains one full oscillation network bounded by half oscillations ready for further composition with other half oscillations.

1.7.8.5. Network expression Figure 1.29e: finest granularity of closure

In Figure 1.29e the completeness and enable behaviors of the link are more finely integrated into the network locality by locality forming the finest granularity longest pipeline version of the network with length of pipeline being characterized by the number of oscillation networks from interaction localities to result locality.

Each **net3** expression contains two links and one complete oscillation network bounded by half oscillations.

```
(net3(A/{1 0}/,in.comp/  B/{1 0}/,in.comp/ => C/{1 0}/,C.omp/)
        (allofrank/{3 2 1 0}/      allofrank.comp/)
        in.comp<=?[allofrank/{3<=[A/1  B/1]
                              2<=[A/1  B/0]
                              1<=[A/0  B/1]
                              0<=[A/0  B/0] }   ~allofrank.comp]
        allofrank.comp<=?[C/{1<={allofrank/1 allofrank/2}
                              0<={allofrank/0 [allofrank/3} } ~C.comp] )
```


The expressed network:

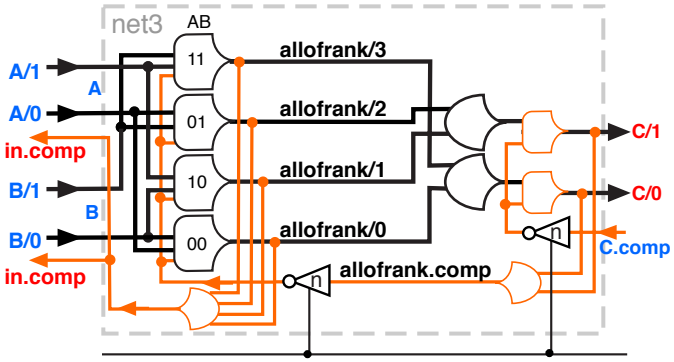


Figure 1.31. net3 network

bignete is composed with three references to **net3**:

```
(bignete(W  X  Y  Z  => C)
              C<=net3(net3(W X) net3(Y Z))  )
```

bignete/C inherits from **net3/C**
bignete/W and **bignete/Y** inherit from **net3/A**
bignete/X and **bignete/Z** inherit from **net3/B**
The inheritance expanded network expression.

```
(bignete(W/{1 0}/,W.comp/  X/{1 0}/,X.comp/
              Y/{1 0}/,Y.comp/  Z/{1 0}/,Z.comp/  => C/{1 0}/,C.comp/)
        C,C.comp<=net3(net3(W,W.comp  X,X.comp) net3(Y,Y.comp  Z,Z.comp)) )
```

bignete is a pipeline network containing four oscillation networks two of which are concurrent and bounded by half oscillations ready for further composition.

1.7.8.6. The collaboration

The closure network coordinates the flow of the interaction network and can affects its throughput, latency and cost but does not affect the interaction behavior itself. All five versions of the interaction network and counter flowing closure networks in Figure 1.29 deliver the same interaction behavior.

1.7.9. The autonomous pipeline

A pipeline network expression is bounded by half oscillations ready for further composition. The bounding half oscillations can close on themselves with their own completeness enlivening and isolating the pipeline network. The color coding represents input and output through binding portals in relation to the expression. Red is output to **rotate4**. **Blue** is input from **rotate4**.


```
( (   =>   )
  ( A/{3 2 1 0}/    B/{3 2 1 0}/   B.comp/   A.comp/)
B.comp<=?B,B.comp<=rotate4(A/{3<=N
                             2<=N
                             1<=N
                             0<=~A.comp } )
        )
```

B.comp is dependent on **B** which is dependent through **rotate4** on A which is dependent on **A.comp**.

1.7.9.1. Auto producing and auto consuming

The differentnesses **A/3**, **A/2** and **A/1** are constant **N**. The conversion of **A.comp** from the **rotate4** reference becomes the differentness presented to **A/0** *auto producing* **D** completeness and completely **N** for interaction locality **A** which alternates between **D** completeness and completely **N**. The completeness of result locality **B.comp** is returned to the **rotate4** reference *auto consuming* the result locality differentness.

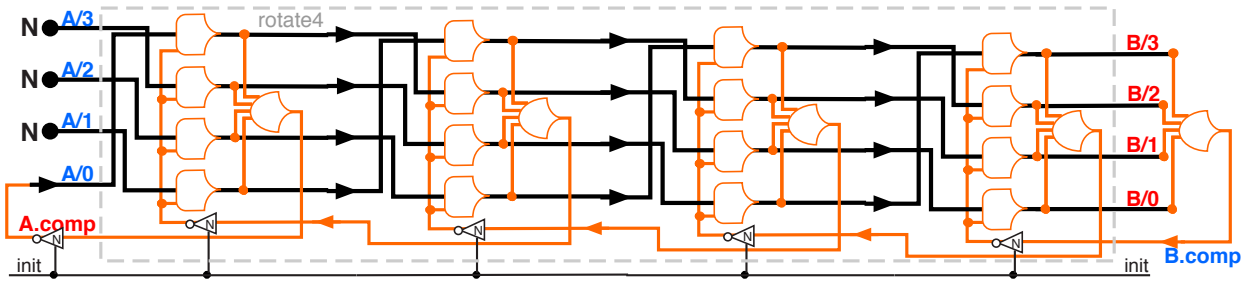


Figure 1.32. Enlivened and isolated pipeline.

The expression has no binding portal. It is not dependent on influence from any extrinsic agency and it does not relate to any extrinsic environment. When **init** is released the pipeline network expressed entirely in terms of the primitive behaviors of section 1.2 begins flowing: feeding itself and emptying itself. The expressed network is isolated, self contained, self timing, self coordinating and autonomously behaving. A behaving universe complete unto itself.

1.7.10. The immersed pipeline

A.comp can control a sensor that samples some aspect of the environment. With **~A.comp/N** the sensor will present a **D** completeness. With **~A.comp/D** the sensor will present completely **N**. The completeness of the interacting locality is still appreciated and the result locality is auto consumed but the differentness of the result locality projects into the environment and imposes its differentness on the environment.

The dependency relations.


```
( (   =>   )
  ( A/{3 2 1 0}/   B/{3 2 1 0}/  sensor/{3 2 1 0}/   B.comp/  A.comp/ )
B.comp<=?B,B.comp<=rotate4([sensor,A.comp   ~A.comp] )
imposedresult<=B
        )
```

imposedresult is dependent on **B** which is dependent through **rotate4** on **sensor** and ~A.comp.

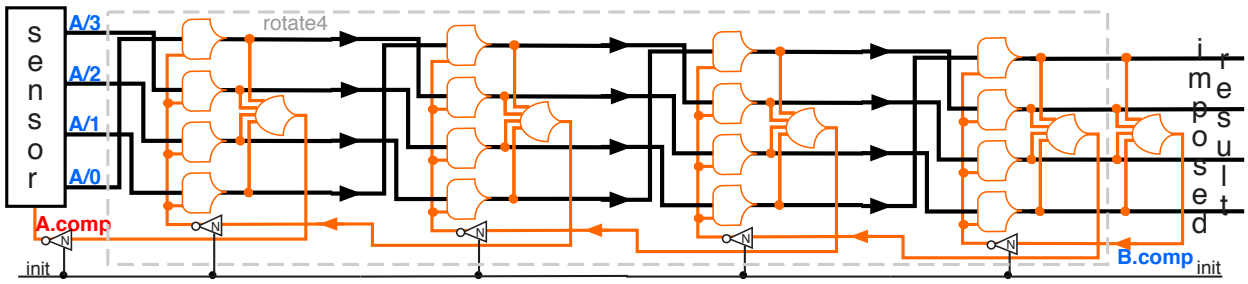


Figure 1.33. The enlivened and engaged pipeline.

There is no binding portal and no extrinsic agency but there is an extrinsic environment. The pipeline is in complete control of its own liveness and its own behavior in relation to the environment in its own space with its own time. The **sensor** may or may not be expressible with the primitives of section 1.2.

1.8. The ring network expression

The other way to isolate and automate a pipeline network is to connect a result locality to an interaction locality forming a ring network.

The pipeline expression **rotate4** of section 1.7.7.2 is closed into a ring.

```
( (   =>   )
  ( B )
B<=rotate4(B)   )
```

B is dependent through **rotate4** on **B** closing the ring.

The locality structures and closures inherited from **rotate4**.

```
( (   =>   )
  ( B/{3 2 1 0}/     B.comp/)
B,B.comp<=rotate4(B,B.comp)   )
```

The expressed network.

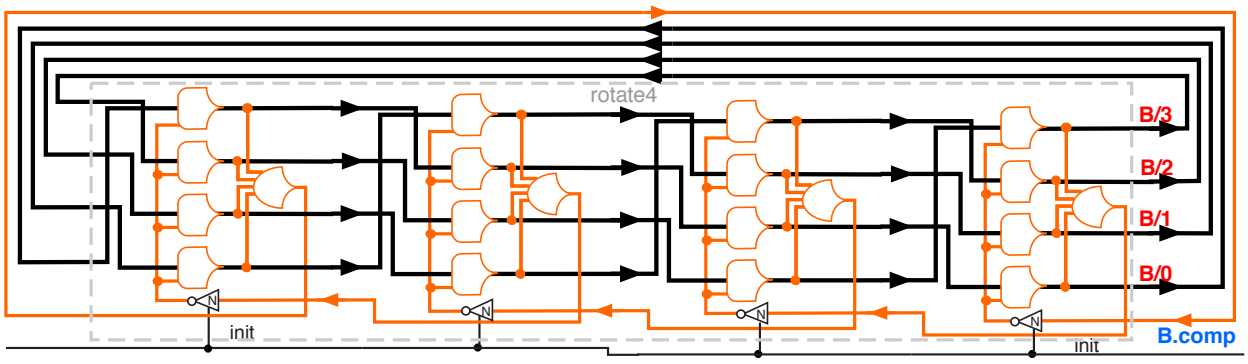


Figure 1.34. rotate4 ring.

1.8.1. Ring initialization

The ring has no wavefronts and with no binding portal cannot receive any wavefronts. There is no completely **N** wavefront to initialize the network and when init is removed there is no **D** completeness wavefront to flow. A **D** completeness wavefront and completely **N** wavefront pair must be initialized in the ring at the time of initialization. To illustrate the details of initialization the ring and its initialization is expressed in terms of half oscillations. One half oscillation expresses a condition shift. Three half oscillations express passage of the locality differentness without change two of which initialize to **D** completeness and to completely **N** wavefronts.

```
( (   =>   )
  ( B:3/{3 2 1 0}/   A/{3 2 1 0}/  C/{3 2 1 0}/   D/{3 2 1 0}/
                     A.comp/  B.comp/    D.comp/    C.comp/ )
A.comp<=?[B/{3<=A/0
                    2<=A/3
                    1<=A/2
                    0<=A/1    ~~B.comp]
B.comp<=?C<=[B  ~C.comp]                    /*  each syntactic term is a half oscillation */
C.comp<=?D<=[C  ~D.comp]
D.comp<=?A<=[D  ~A.comp]   )
```

B is dependent on **A** which is dependent on **D** which is dependent on **C** which is dependent on **B** closing the ring.

1.8.1.1. The initialization protocol

Locality **B** is initialized to **D** completeness **3** with **B:3**. For any locality initialized to a **D** completeness wavefront all of its dependent localities must be initialized to a completely **N** wavefront to block the **D** conditions of the **D** completeness initialization from progressing and to establish the completely **N** wavefront that will initialize the rest of the expressed network to completely **N**. The conversion closing with the initialized locality, **~~B.comp**, is not initialized to **N** to allow it to transition to **D** to retain the initialized **D** completeness wavefront and to block the initializing completely **N** wavefront when it reaches the initialized locality.

This is the protocol for any **D** completeness wavefront initialization specified by **localityname:differentness**.

The expressed network:

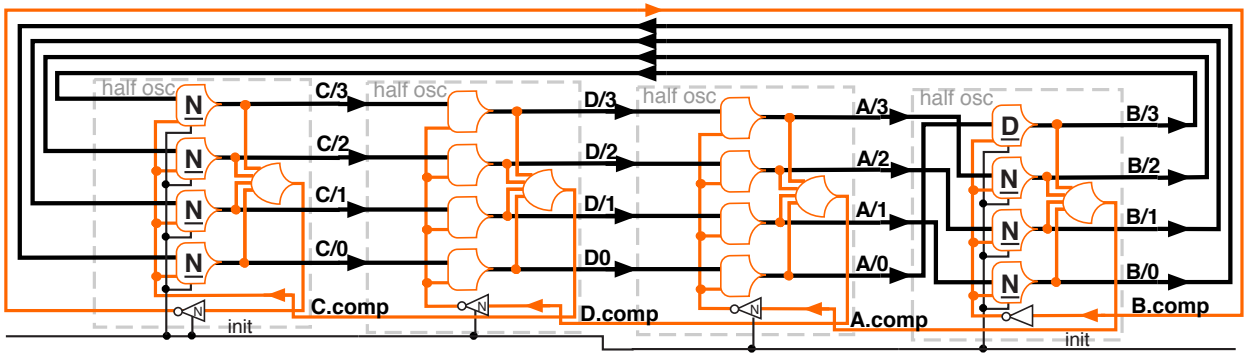


Figure 1.35. The ring with B initialized to B/3.

When initialization is released the initialized conversions transition to **D** allowing the initialized **D** completeness wavefront to begin flowing which will continue flowing indefinitely around the ring followed by a completely **N** transition wavefront.

1.8.1.2. Initialization pipeline segment

An initialization can be expressed with a pipeline segment and referenced to incorporate the initialization into a larger expression.

```
(init3(A/(3 2 1 0),A.comp/ => B/(3 2 1 0),B.comp/)
  ( C:3/(3 2 1 0)/    C.comp/ )
  C.comp<=?B<=[C  ~B.comp]
 A.comp<=?C<=[A  ~~C.comp]  )
```

The expressed network.

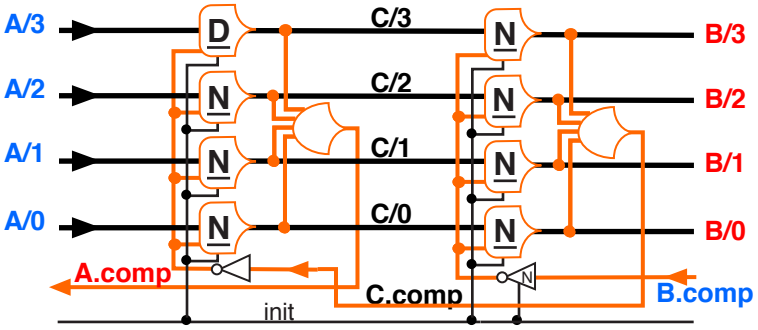


Figure 1.36. init3 initialize pipeline segment.

1.8.2. The source ring

An indefinitely flowing ring can become a source of wavefront flow expressed with a binding portal with only a result locality. With a binding portal the expression can be referenced

to access its wavefronts. **sourceB** composed with **init3** and **rotate** from section 1.7.7. delivers a shift of differentness by one each ring cycle.

When ~**Z.comp** transitions to **D** and the ring delivers a **D** completeness wavefront. When ~**Z.comp** transition to **N** and the ring deliver a completely **N** wavefront.

Source ring expression:

```
(sourceB(   => Z )
  ( B )
  Z<=B
  B<=(init3( rotate( B ) )   )
```

B depends through **init3** and through **rotate** on **B** closing the ring.

The full expression with locality structures and closures inherited from **rotate** and **init3**.

```
(sourceB(   => Z/{3 2 1 0}/,Z.comp/ )
  ( B/{3 2 1 0}/     B.comp/ )
  Z<=B
  B,[B.comp  Z.comp]<=(init3( rotate( B,B.comp ))   )
```

The expressed network:

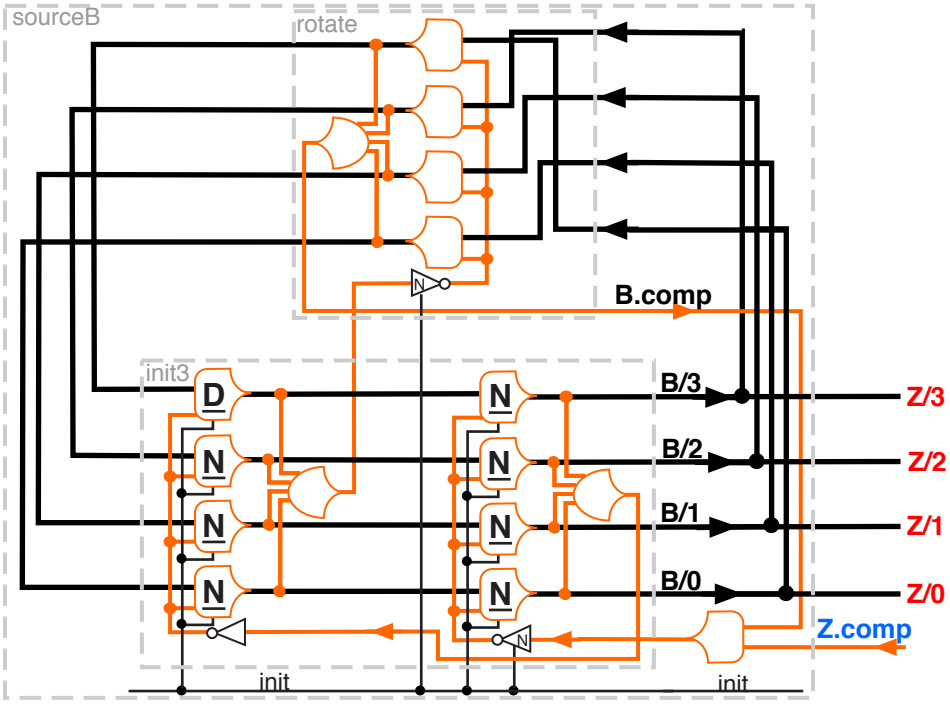


Figure 1.37. Source ring network.

B now flows to two dependencies and is closed from both dependencies.

A source ring can deliver a progression of differentnesses such as random numerals, Fibonacci numerals, counting numerals, cyclic control tokens and so forth. It is referenced as:


```
sourceB( => X))                    /* full portal */

X<=sourceB( )                      /* locality nesting */

expressionreference( sourceB( ) )  /* reference nesting */
```

1.8.3. The interaction ring

A ring with a binding portal that has an interaction locality as well as a result locality.

The dependency expression:

```
(behaviorB(A/{1 0}/ => Z/{3 2 1 0}/)
  ( B/{3 2 1 0}/ C/{3 2 1 0}/ D/{3 2 1 0}/ )
  Z<=B
  init3( C => B)
  rotate( B => D)
        C/{3<={[A/1 D/3] [A/0 D/2]}
           2<={[A/1 D/2] [A/0 D/1]}
           1<={[A/1 D/1] [A/0 D/0]}
           0<={[A/1 D/0] [A/0 D/3]} } )
```

The interaction dependency relations determine the locality structures of the interacting localities which must match the inheritances from the binding portal references.

Z is dependent on B which is dependent through **init3** on **C** which is dependent on **A** and on **D** which is dependent on **B** closing the ring. **A/0** undoes the rotate on the **D** locality. **A/1** passes the rotated **D** locality.

The inheritance expression:

```
(behaviorB(A/{1 0}/,C.comp/ => Z/{3 2 1 0}/,Z.comp/ )
( B/{3 2 1 0}/ C/{3 2 1 0}/ D/{3 2 1 0}/ B.comp/ C.comp/ D.comp/ )
  Z<=B
 init3( C,C.comp => B,[Z.comp B.comp] )
  rotate( B,B.comp => D,C.comp)
        C/{3<={[A/1 D/3] [A/0 D/2]}
           2<={[A/1 D/2] [A/0 D/1]}
           1<={[A/1 D/1] [A/0 D/0]}
           0<={[A/1 D/0] [A/0 D/3]} } )
```

The expressed network.

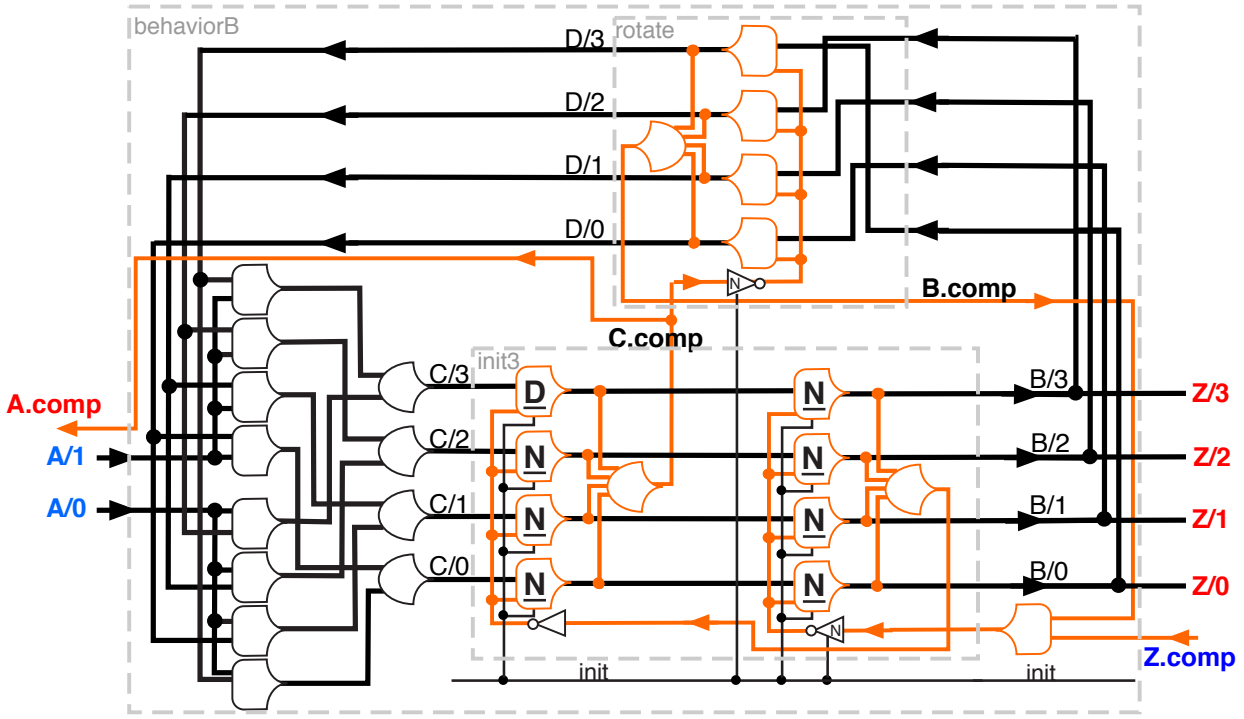


Figure 1.38. Ring with exposed binding portal.

1.8.3.1. Loss of constancy in relation to a binding portal

While the intrinsic interaction dependency relations might be constant in relation to both the ring feed back differentness **D** and the presented differentness **A** of the binding portal interaction locality it is not constant in relation to the binding portal interaction locality alone. The feed back from the ring forms an intrinsic memory of the previous result within the expression which influences the interaction result which does not always assert the same result differentness for the same differentness presented by the binding portal interaction locality.

An interaction can be constant in relation to a binding portal only when the result locality is dependent on all and only of the interaction localities presented by the binding portal.

1.8.4. Composing rings

The ring becomes the new unit of composition. Rings can be composed by connecting through a link or by sharing a pipeline segment.

1.8.4.1. Composing rings by linking half oscillations

Expression **alternate** is a source ring that can supply wavefronts of alternating differentness.


```
(alternate( => Z/{1 0},Z.comp/)
        ( A/{1 0}/ B/{1 0}/ C:1/{1 0}/   A.comp/  B.comp/  C.comp/ )
        Z<=A
        A.comp<=?[A/{1<=B/0
                        0<=B/1 }  ~[C.comp  Z.comp] ]
        B.comp<=?B<=[C  ~A.comp]
        C.comp<=?C<=[A  ~~B.comp]
    )
```

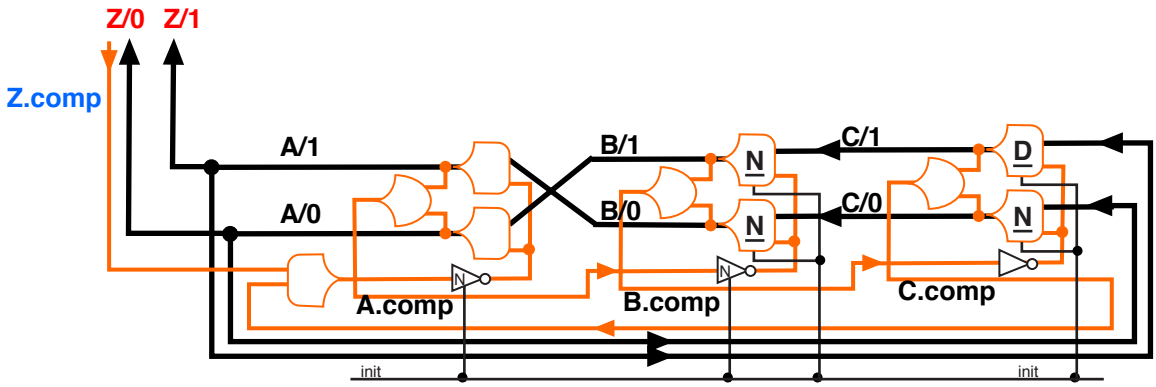


Figure 1.39. Source ring alternating differentness.

The source ring locality **alternate/Z** is linked to locality **behaviorB/A** making the binding portal interaction locality of **behaviorB** intrinsic.

Dependency relations:

```
(bigsourceB( => B )
        behaviorB(alternate( ) => B) )
```

B depends on **behaviorB** which depends on **alternate**.

The full expression with locality structures and closures inherited from **behaviorB** and **alternate**.

```
(bigsourceB( => B/{3 2 1 0}/,B.comp )
         behaviorB(alternate( ) => B,B.comp) )
```

The expressed network.

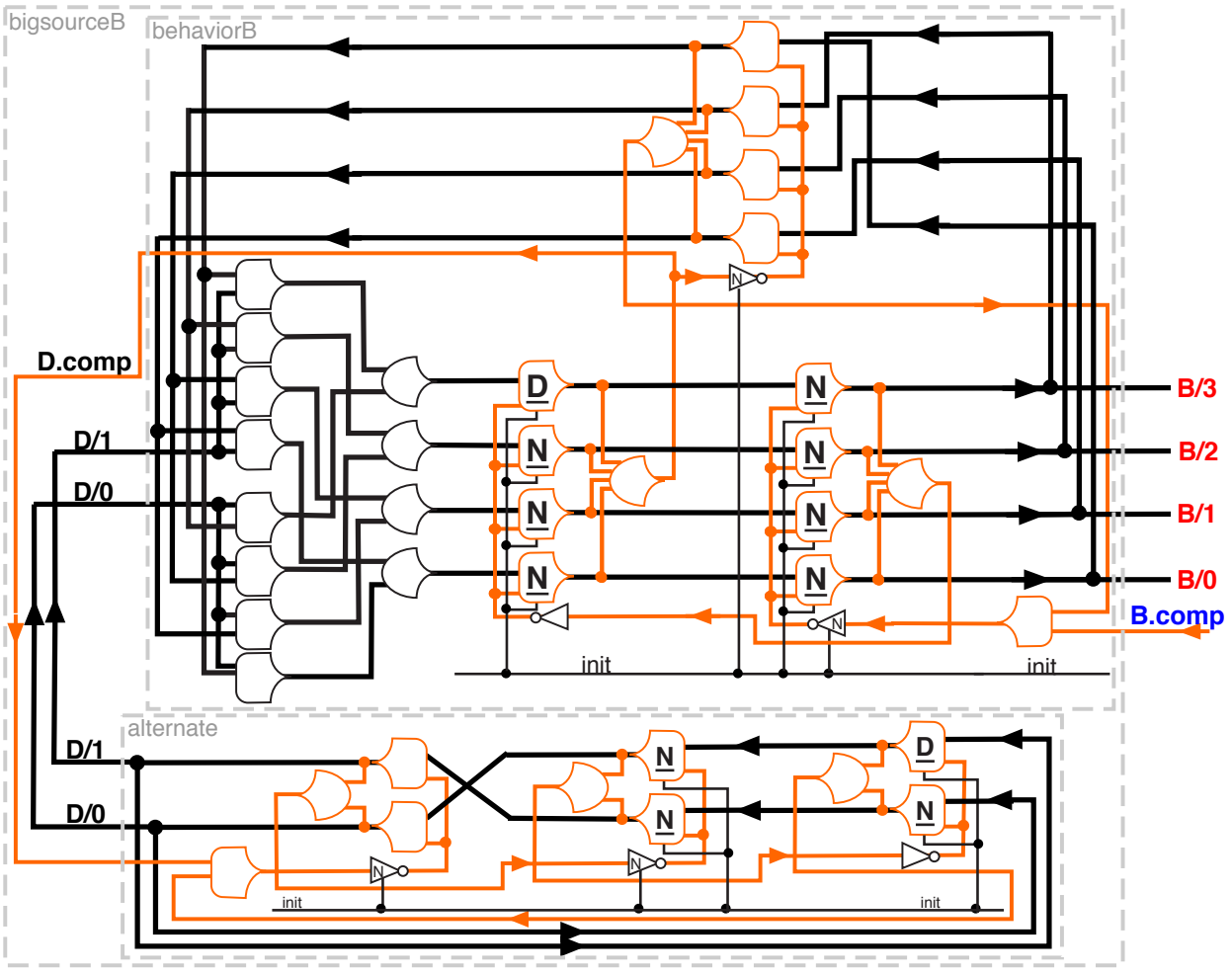


Figure 1.40. The network of linked rings

The binding portal interaction locality is no longer extrinsically exposed but is intrinsically incorporated into a greater network behavior. The two expressions honor each others flow behavior. The ring **alternate** transitions to **D** completeness only after the transition of **D.comp** to **N**. Ring **alternate** transitions to completely **N** only after the transition of **D.comp** to **D**.

1.8.4.2. Composing rings by sharing pipeline network segments

Rings can be composed by coupling the rings, not through their binding portals, but directly in the dependency expression by sharing one or more pipeline segments. In **bigsourceC** a half oscillation pipeline segment is coupled into a ring expression to form a second coupled ring.

Dependency expression:


```
(bigsourceC(   =>   Z)
( A/{1 0}/   B/{3 2 1 0}/  )
Z<=B
behaviorB( A  => B )
B.comp<=?[A/{1<={B/1 B/3}                    /*  the coupled half oscillation */
                2<={B/0 B/2} }   A.comp ]
  )
```

Z depends on **B** which depends through **behaviorB** on **A** which depends through the half oscillation on **B** closing the coupled ring. The half oscillation coupled through the binding portal of **behaviorB** forms a ring from its result locality to its interaction locality unexposing the interaction locality of the **behaviorB** binding portal.

The full expression with locality structures and closures inherited from **behaviorB**.

```
(bigsourceC(   =>   Z/{1 0},Z.comp/)
( A/{1 0}/   B/{3 2 1 0}    A.comp/   B.comp/  )
Z<=B
behaviorB( A,A.comp  => B,[B.comp  Z.comp])
B.comp<=?[A/{1<={B/1 B/3}                    /* the  coupled half oscillation */
                0<={B/0 B/2} }   ~A.comp ]
  )
```

The expressed network:

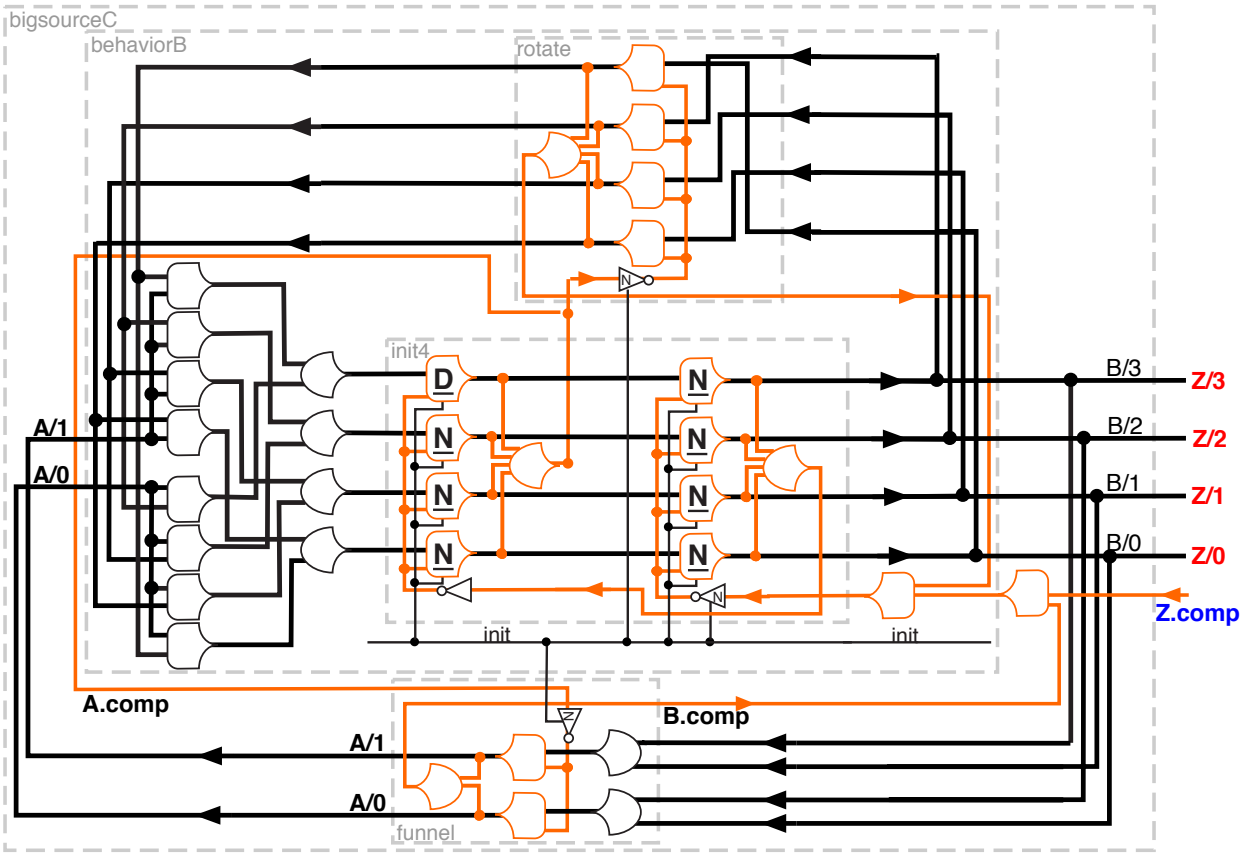


Figure 1.41. Coupled rings.

The half oscillation network can be expressed as **funnel** and coupled by reference.

```
(funnel(B,B.comp => A,A.comp)
B.comp<=?[A/{1<={B/1 B/3}
                0<={B/0 B/2} }  ~A.comp ]
  )
```

and its binding portal referenced to express **bigsourceC** with two coupled rings.

```
(bigsourceC(  =>  Z/{1 0},Z.comp/)
( A/{1 0}/   B/{3 2 1 0}   A.comp/    B.comp/ )
Z<=B
behaviorB( A,A.comp  => B,[B.comp  Z.comp] )
funnel( B,B.comp => A,A.comp )
  )
```

1.9. The LFSR source network of coupled rings

The **LFSR** (Linear Feedback Shift Register) is a network of 5 rings with 6 independently flowing wavefronts coupled through a structure of shared pipeline segments. Each wavefront

represents a different instance of interaction time with 5 instances simultaneously interacting through a shared interaction network.

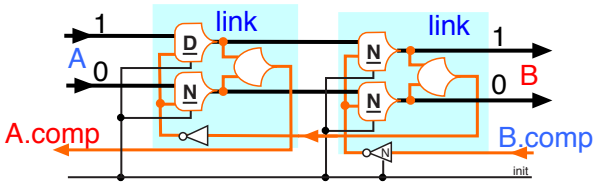
The **LFSR** is a tapped delay pipeline which taps present to a network of **XOR** behaviors the result of which presents to the tapped delay pipeline closing the ring networks and which is the result of the **LFSR**.

1.9.1. The tapped delay pipeline

The **LFSR** tapped delay pipeline is composed of of six pipeline segments five of which initialize a wavefront to **1 - init1** or to **0 - init0**.

init1 pipeline segment

```
(init1(A/{1 0}/,A.comp/ => B/{1 0}/,B.comp/)
        ( D:1/{1 0}/   D.comp/ )
        A.comp<=?D<=[A  ~D.comp]
        D.comp<=?B<=[D  ~B.comp]  )
```



init0 pipeline segment

```
(init0(A/{1 0}/,A.comp/ => B/{1 0}/,B.comp/)
        ( D:0/{1 0}/   D.comp/ )
        A.comp<=?D<=[A  ~D.comp]
        D.comp<=?B<=[D  ~B.comp]  )
```

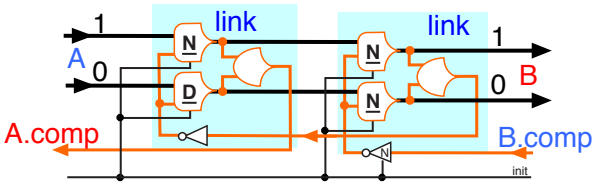


Figure 1.42. Initializing pipeline segments.

The delay pipeline is initially expressed with reference nesting with interaction locality **O** and result locality **F** :

```
F<=init1(init0(init1(init0(init0(init0( O ))))))
```

To tap localities of the pipeline specific pipeline segment references of the pipeline are assigned result locality names, **B**, **C**, **D** and **E** and are re-expressed with locality nesting within the reference nesting.

```
F<=init1(E<=init0(D<=init1(C<=init0(B<=init0(init0( O ))))))
```

The result locality names are now visible and can be referenced with name correspondence by the **XOR** behaviors.

1.9.2. XOR network for LFSR

The **XOR** network is a network of four **XOR** half oscillations.

The **XOR** half oscillation.


```
(XOR(A/{1 0}/,AB.comp/  B/{1 0}/,AB.comp/ =>
                                C/{1 0}/,C.comp/)

AB.comp<=?[C/{1<={[A/0  B/1]  [A/1  B/0]}
                0<={[A/0  B/0]  [A/1  /B/1]} }
                                ~C.comp] )
```

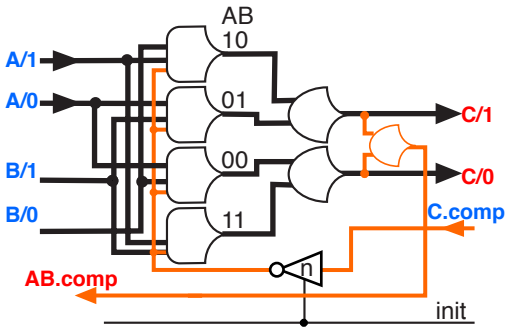


Figure 1.43. The XOR half oscillation.

The **XOR** network expressed with reference nesting of the **XOR** half oscillations each referencing its interaction localities by name correspondence from the result locality taps of the delay pipeline.

```
O<=XOR(F  XOR(XOR(B C)  XOR(D E)))
```

Locality **O** of the pipeline network is substituted with **XOR** network which produces locality **O**. The locality nesting expression of the **XOR** network is retained because **O** is referenced to express the **LFSR** result.

The dependency expression of the **LFSR**:

```
F<=init1(E<=init0(D<=init1(C<=init0(B<=init0(init0(O
            <=XOR(F  XOR(XOR(B C)  XOR(D E)))))))))
```

The dependency expression is a single syntax structure with syntactically remote places in the structure connected by name correspondence. Name correspondence can express connections that syntax relations cannot express.

The complete expression of the **LFSR** network:

```
(LFSR (  => out  )    (B  C  D  E  F  O)
    F<=init1(E<=init0(D<=init1(C<=init0(B<=init0(init0(O
            <=XOR(F  XOR(XOR(B C)  XOR(D E)))))))))
    out<=O  )
```

locality out is dependent on **O** which is dependent through the **XOR** pipeline on **F**, **E**, **D**, **C** and **B** each of which are dependent on **O** closing all five coupled rings.

The locality structure inheritance:

```
(LFSR (  => out  )    (B/{1 0}/  C/{1 0}/  D/{1 0}/  E/{1 0}/  F/{1 0}/  O/{1 0}/)
    F<=init1(E<=init0(D<=init1(C<=init0(B<=init0(init0(O
            <=XOR(F  XOR(XOR(B C)  XOR(D E)))))))))
    out<=O  )
```


The expression cannot directly represent closure inheritance because in the locality nesting within reference nesting **init0(D<=init1)** the single reference to **D** represents both the result locality of one binding portal and an interaction locality of another binding portal which may have different closure references. The inheritance of closure will be further discussed in section 1.9.3.

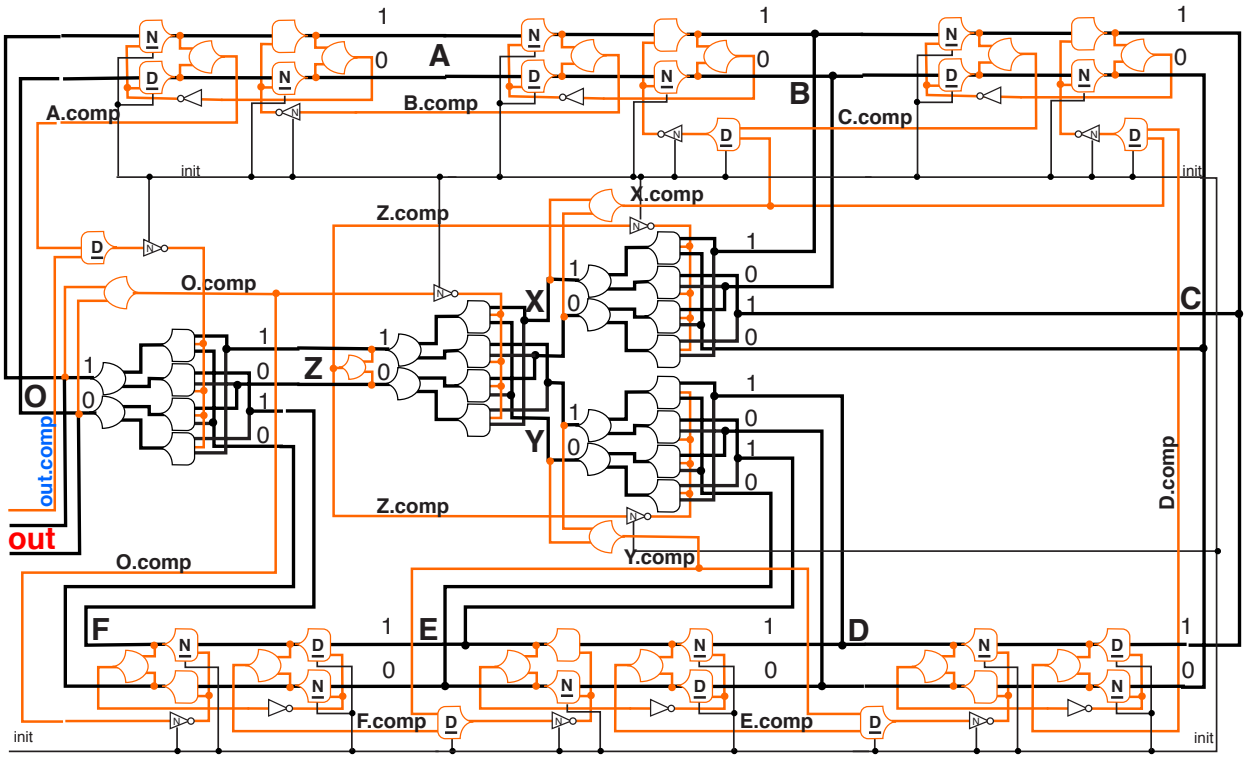


Figure 1.44. LFSR source network of multiple coupled rings.

The **LFSR** network contains 6 different initialized wavefronts continually flowing around the delay pipeline representing 6 differentnesses of time 5 of which simultaneously interact through the shared **XOR** interaction network.

The 5 rings are color coded in Figure 1.45 to highlight their sharing of segments of the delay pipeline. All five rings share the **XOR** pipeline.

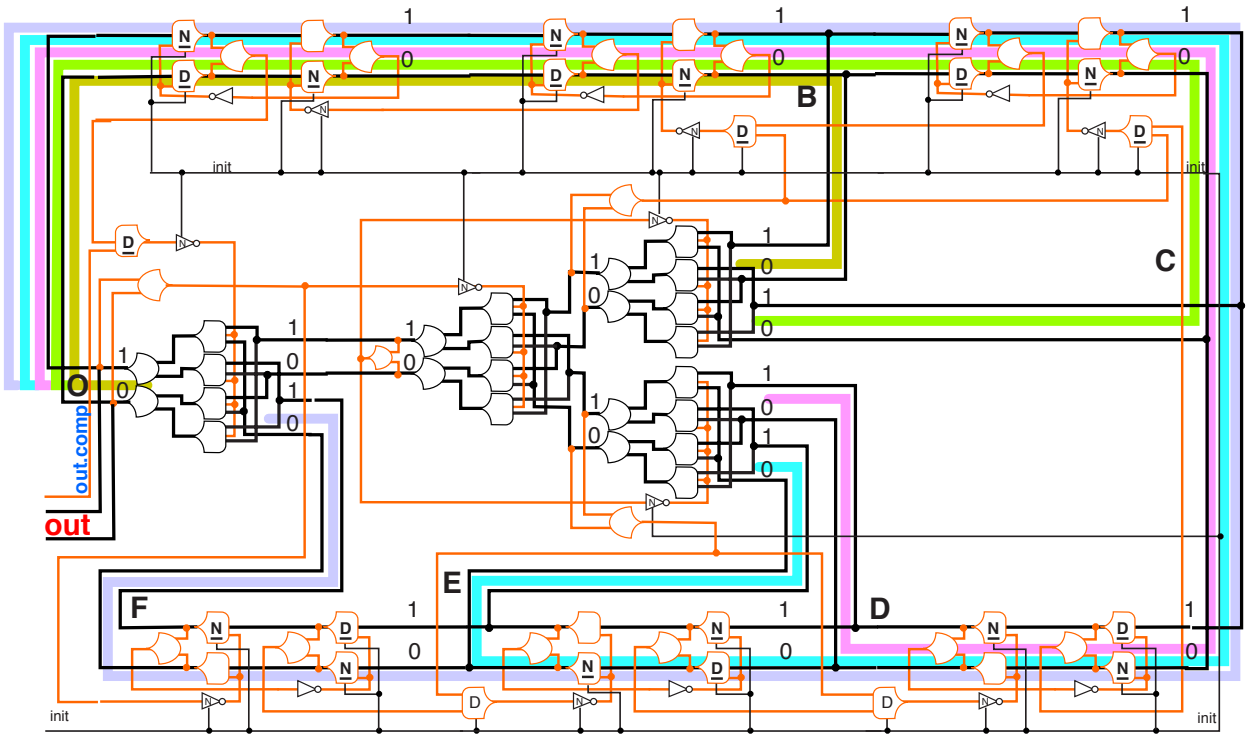


Figure 1.45. Coupled rings and their shared pipeline components.

1.9.3. Language interlude

The **LFSR** expression provides an opportunity for a deeper discussion of the collaboration relationships in a dependency language expression between syntax structure and name correspondence, between interaction and closure and among binding portals.

1.9.3.1. Maximal syntax and minimal name correspondence

```
(LFSR (  => out  )    (B  C  D  E  F  O)
        F<=init1(E<=init0(D<=init1(C<=init0(B<=init0(init0(O
                    <=XOR(F  XOR(XOR(B C)  XOR(D E)))))))))
        out<=O  )
```

The above **LFSR** expression is a conveniently short and compact expression of dependency relations in terms of maximal syntax expression and minimal name correspondence expression. With syntax nesting dependency relations in the XOR syntactic nestings the involved localities are not explicitly expressed.

1.9.3.2. Minimal syntax and maximal name correspondence

A minimal syntax expression is derived by backing out all syntax nesting relations and express all of the binding portal reference as full portal references. The binding portal reference

which must match the referenced binding portal is a minimal unit of syntactic differentness expression that cannot be made any smaller or simpler. A minimal syntax expression is in terms of individual binding portal references, with no syntactic reference nesting eliciting the the explicit reference of all locality names and their correspondence relations to connect the individual syntax structures. It will be necessary to assign new names to previously unnamed localities that were nested such as **X**, **Y**, **Z** and **A** below. In the expression below **red** is output in relation to **LFSR** and **blue** is input in relation to **LFSR**.

The minimal syntax expression of **LFSR**:

```
(LFSR   (  => out )  ( A  B  C  D   E  F  O  X  Y  Z )
              init1( E  =>  F )
              init0( D  =>  E )
              init1( C  =>  D )
              init0( B  =>  C )
              init0( A  =>  B )
              init0( O  =>  A )
              XOR( F  Z  =>  O )
              XOR( X  Y  =>  Z )
              XOR( B  C  =>  X )
              XOR( D  E  =>  Y )
              out<=O          )
```

1.9.3.3. Sameness, differentness and specificity

The only names not arbitrarily chosen in a minimal syntax expression are the reference names of the referenced binding portals. All of the locality names are arbitrary chosen. Each locality name is different but it is the sameness relations (correspondences) of the locality name references connecting different syntax structures and the sameness of each syntax structure connecting different names that expresses the specific structure of interaction dependency relations.

1.9.3.4. Deriving closure

The structure of closure is a dual complement of and can be derived from the structure of the interaction dependency relations. Each link locality is closed by the completeness of all the interaction localities to which it contributed completeness and closes with all of the localities which contributed to its completeness (see section 1.7.6.2).

In the expression below each result locality closes with all of the localities which contributed to its completeness which are all of the interaction locality references of its binding portal reference.


```
(LFSR   (  => out )  ( A  B  C  D  E  F  O  X  Y  Z )
A.comp/  B.comp/  C.comp/  D.comp/  E.comp/
                        F.comp/  O.comp/  X.comp/  Y.comp/  Z.comp/  )
                  init1( E,F.comp  =>  F )
                  init0( D,E.comp  =>  E )
                  init1( C,D.comp  =>  D )
                  init0( B,C.comp  =>  C )
                  init0( A,B.comp  =>  B )
                  init0( O,A.comp  =>  A )
                  XOR( F,O.comp  Z,O.comp  =>  O )
                  XOR( X,Z.comp  Y,Z.comp  =>  Z )
                  XOR( B,X.comp  C,X.comp  =>  X )
                  XOR( D,Y.comp  E,Y.comp  =>  Y )
                  out<=O          )
```

In the expression below each result locality is closed by all of the interaction localities to which it contributed completeness.

```
(LFSR   (  => out )  ( A  B  C  D  E  F  O  X  Y  Z )
A.comp/  B.comp/  C.comp/  D.comp/  E.comp/
                        F.comp/  O.comp/  X.comp/  Y.comp/  Z.comp/  )
                  init1( E,F.comp  =>  F,O.comp )
                  init0( D,E.comp  =>  E,[F.comp  Y.comp] )
                  init1( C,D.comp  =>  D,[E.comp  Y.comp] )
                  init0( B,C.comp  =>  C,[D.comp  X.comp] )
                  init0( A,B.comp => B,[C.comp  X.comp] )
                  init0( O,A.comp => A,B.comp )
                  XOR(F,O.comp  Z,O.comp => O,[A.comp  out.comp] )
                  XOR(X,Z.comp  Y,Z.comp => Z,O.comp )
                  XOR(B,X.comp  C,X.comp => X,Z.comp )
                  XOR(D,Y.comp  E,Y.comp => Y,Z.comp )
                  out<=O )
```

In the expression below each locality inherits its d=structure from referenced binding portals presenting a fully explicit minimal syntax expression of the **LFSR** is:


```
(LFSR    (  => out/{1 0}/,out.comp/ )
( A/{1 0}/  B/{1 0}/  C/{1 0}/  D/{1 0}/  E/{1 0}/  F/{1 0}/  O/{1 0}  X/{1 0}/  Y/{1 0}/  Z/{1 0}/
 A.comp/  B.comp/  C.comp/  D.comp/  E.comp/
                        F.comp/  O.comp/  X.comp/  Y.comp/  Z.comp/  )
        init1( E,F.comp  =>  F,O.comp)
        init0( D,E.comp  =>  E,[F.comp  Y.comp] )
        init1( C,D.comp  =>  D,[E.comp  Y.comp] )
        init0( B,C.comp  =>  C,[D.comp  X.comp] )
        init0( A,B.comp => B,[C.comp  X.comp] )
        init0( O,A.comp => A,B.comp )
        XOR(F,O.comp  Z,O.comp => O,[A.comp  out.comp]  )
        XOR(X,Z.comp  Y,Z.comp => Z,O.comp )
        XOR(B,X.comp  C,X.comp => X,Z.comp )
        XOR(D,Y.comp  E,Y.comp => Y,Z.comp )
        out<=O )
```

Again, the direction coloring is in relation to the **LFSR** expression.

This minimal syntax expression straightforwardly expands with expression inheritance to an expression entirely in terms of primitive behaviors which maps directly to the dependency network of Figure 1.44.

1.9.3.5. The intersecting dependency flows

The interaction dependency relations and the closure dependency relations weave the binding portal references into a self coordinating fabric of computation. The left of Figure 1.46 shows the dependency flow of interaction localities through the text of expression expressed by connecting corresponding locality names. The right shows the dependency flow of closure relations expressed by connecting corresponding closure names. Notice that in both cases all dependency flow is from blue to red.

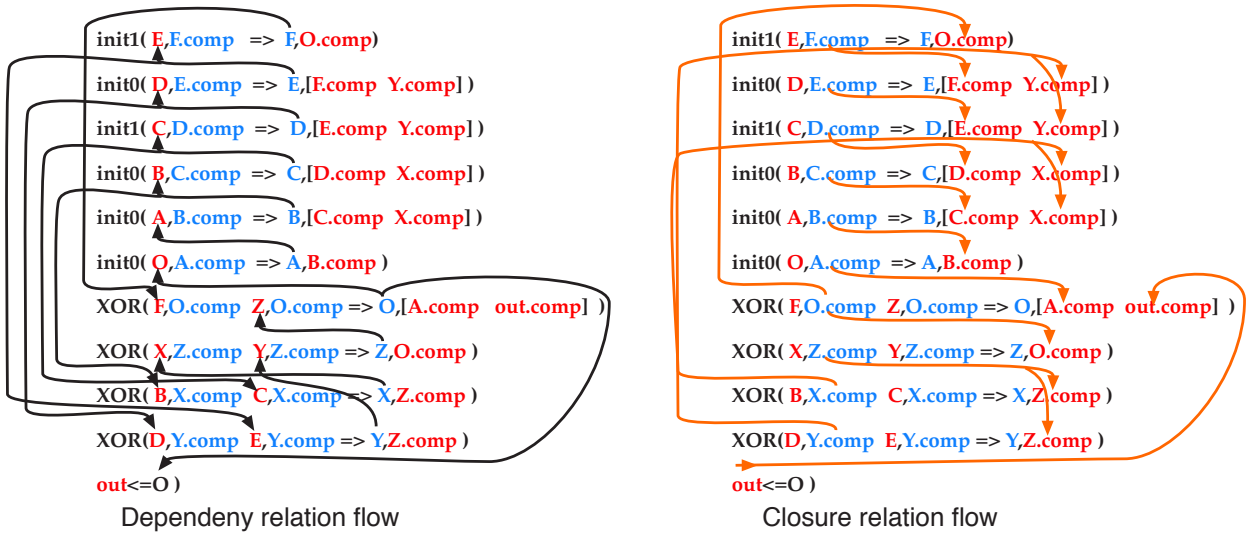


Figure 1.46. LFSR flow of dependency relations and flow of closure relations.

Figure 1.47 shows the graphical structure of the link locality dependency relations for both the interaction dependency relations and the closure dependency relations. Notice that the graph structures are duals with dependency relations flowing in opposite directions.

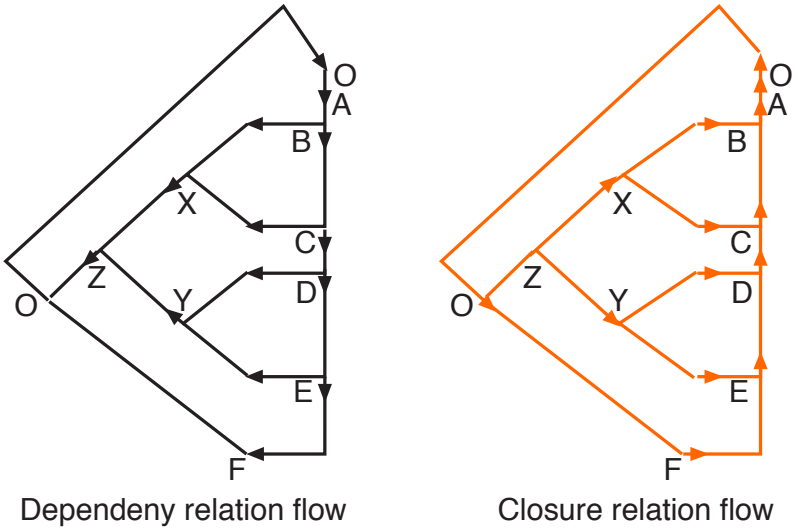


Figure 1.47. Graph representation of the flow relations.

1.9.3.5.1. Sequencing

A minimal syntax expression behavior can also be realized with sequence control rather than with closure coordination by mapping each locality name to an addressable memory location, replacing each locality name reference in the expression with memory address references and arranging the binding portal references in an appropriate sequence such that each memory location (locality) is written before its dependencies read it. The expression under each binding portal reference has its own minimal syntax expression of dependency relations which can be sequenced and so on to ultimately specify a sequence of primitive behaviors and memory address references to a sequential interpreter of primitive behaviors.


```
(LFSR    (  => out/{1 0}/  )
( A:0/{1 0}/  B:0/{1 0}/  C:0/{1 0}/  D:1/{1 0}/  E:0/{1 0}/  F:1/{1 0}/  O/{1 0}  X/{1 0}/  Y/{1 0}/  Z/{1
0}/ )
0       XOR( B  C  =>  X )
1       XOR( D  E  =>  Y )
2       XOR( X  Y  =>  Z )
3       XOR( F  Z  =>  O )
4       out<=O )
5       init0( O  =>  A )       */  enter here  /*
6       init0( A  =>  B )
7       init0( B  =>  C )
8       init1( C  =>  D )
9       init0( D  =>  E )
10      init1( E  =>  F )
11      go to 0
```

This is a partial example to give a sense of how it works. The full sequential derivation is somewhat more complicated requiring unconditional jumping, conditional branching and separating the initialization behaviors. See “A Journey Through Computation” section 4.12.1.

1.9.3.5.2. Recovering the maximal syntax expression

The maximal syntax expression is the smallest and expressionally densest form of expression which might make it a most convenient form expression to initially specify dependency relations. The maximal syntax expression can be recovered from the minimal syntax expression by the transforming full portal references to syntactic nesting references by substituting, everywhere possible, each locality interaction reference with the binding portal reference delivering the locality result.

Beginning with the minimal syntax expression:

```
(LFSR    (  => out )  ( A  B  C  D  E  F  O  X  Y  Z )
        init1( E  =>  F )
        init0( D  =>  E )
        init1( C  =>  D )
        init0( B  =>  C )
        init0( A  =>  B )
        init0( O  =>  A )
        XOR( F  Z  =>  O )
        XOR( X  Y  =>  Z )
        XOR( B  C  =>  X )
        XOR( D  E  =>  Y )
        out<=O          )
```

Locality result references **Z**, **Y**, **X, F** and **A** each associate to a single interaction reference and are substituted directly:


```
(LFSR   (  => out )  ( B  C  D   E  O )
        init0( D  =>  E )
        init1( C  =>  D )
        init0( B  =>  C )
        init0( init0( O )  =>  B )
        XOR( init1( E )  XOR( XOR( B  C )  XOR( D  E ) )  =>  O )
                out<=O          )
```

The remaining localities, each of whose result locality associates to two interaction references, must retain their result locality names to express the fan out associations with name correspondence. At this point it is most straightforward to convert the remaining full binding portal references to locality nesting and then perform the substitution.

Conversion to locality nesting:

```
(LFSR   (  => out )  ( B  C  D   E  O )
                E<=init0( D )
                D<=init1( C )
                C<=init0( B )
                B<=init0( init0( O ) )
                O<=XOR( init1( E )  XOR( XOR( B  C )  XOR( D  E ) ))
                out<=O          )
```

For each locality substitute one of the locality interaction references with the syntax expression providing its result:

```
(LFSR   (  => out )  ( B  C  D   E  O )
                E<=init0( D<=init1( C<=init0( B<=init0( init0( O
                        <=XOR( init1( E )  XOR( XOR( B  C )  XOR( D  E ) )) ) ) ) ) )

                out<=O          )
```

or another of many ways of performing the substitutions:

```
(LFSR   (  => out )  ( B  C  D   E  O )
                O<=XOR( init1( E )  XOR( XOR( B<=init0( init0( O ) )  C<=init0( B ) )
                                         XOR( D<=init1( C )  E<=init0( D ) ) ))
                out<=O          )
```

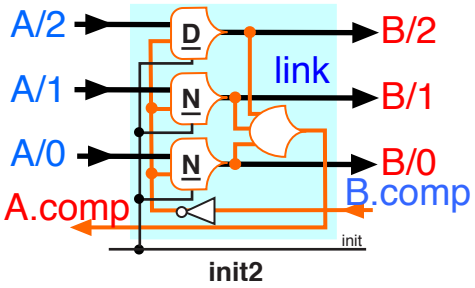
1.10. The immersed ring network: engaging the environment

The immersed ring network of Figure 1.48 is a network of two coupled rings with an imposition portal to influence the environment and a sampling portal sensitive to the effects of the imposition on the environment forming an interaction network capable of remembering and learning.

It is expressed with the component networks below.

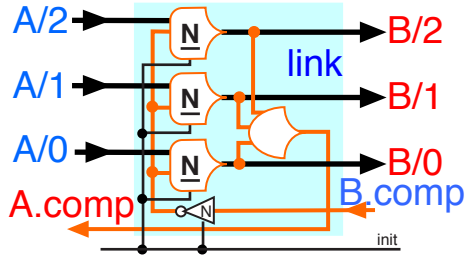
1.10.1. Half oscillation initialization component networks

```
(init2(A/{2 1 0}/,A.comp/ =>
                B:2/{2 1 0}/,B.comp/ )
        A.comp<=?B<=[A  ~B.comp]  )
```



init2

```
(initN(A/{2 1 0}/,A.comp/ =>
                B:N/{2 1 0}/, B.comp/)
        A.comp<=?B<=[A  ~B.comp]  )
```

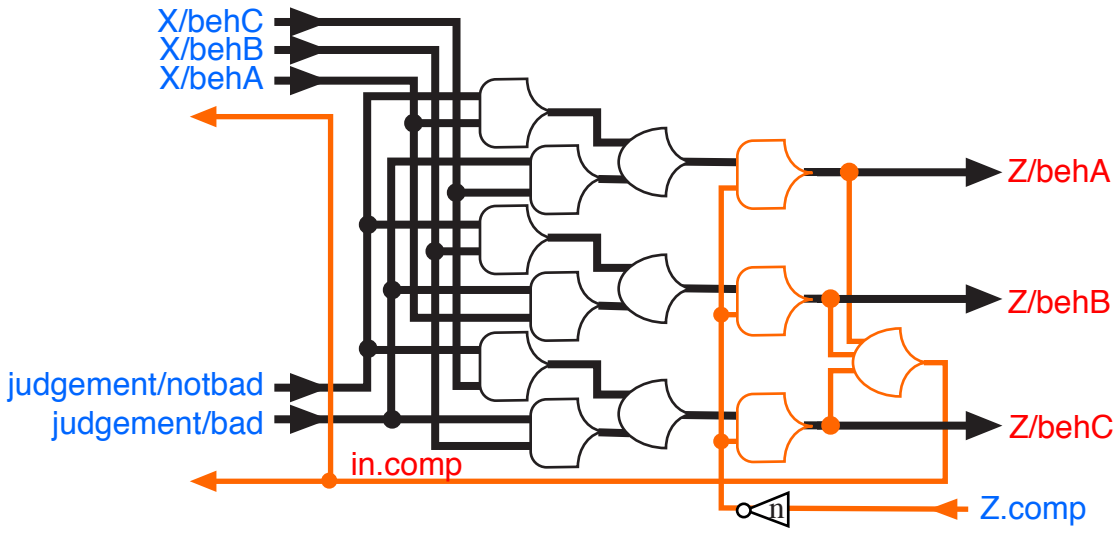


initN

1.10.2. Half oscillation selection component network

There are three network behaviors **behA**, **behB** and **behC**. **judgement/notbad** selects the remembered previous behavior. **judgement/bad** selects the next behavior in rotation.

```
(behmod(X/{behA  behB  behC}/,Z.comp  judgement/{bad  notbad}/,Z.comp =>
                                                Z/{behA  behB  behC}/,Z.comp)
Z.comp<=?Z/{behA<=[{[X/behC  judgement/bad]
                    [X/behA  judgement/notbad]}  ~Z.comp]
            behB<=[{[X/behA  judgement/bad]
                    [X/behB  judgement/notbad]}  ~Z.comp]
            behC<=[{[X/behB  judgement/bad]
                    [X/behC  judgement/notbad]}  ~Z.comp]
                }  )
```

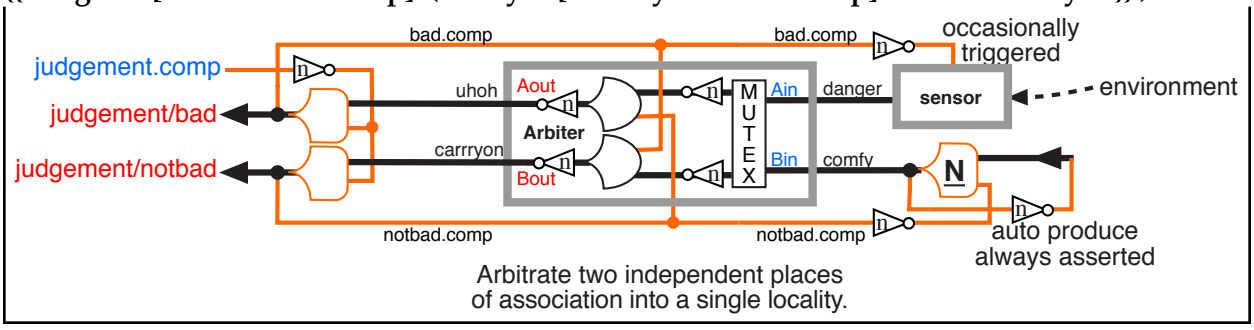
1.10.3. The arbitration pipeline component network

The arbitration component arbitrates the continual liveness flow **comfy** with the occasional sensor flow **danger** into the network as a locality **judgement** of two mutually exclusive differentnesses. Arbitration with a MUTEX is a primitive behavior (see section 1.2.3).

{{Ain Bin => Aout Bout}}

The arbitration expressison:

(next(=> judgement/{bad notbad}/,judgement.comp/)
(carryon/ uhoh/ comfy/ danger/ bad.comp/ notbad.comp/)
bad.comp<=judgement/bad<=[uhoh ~judgement.comp]
notbad.comp<=judgement/notbad<=[carryon ~judgement.comp]
{{danger<=[sensor ~bad.comp] (comfy<=[~comfy ~notbad.comp] => uhoh carryon}})



The dependency relations: **judgement** is dependent through the **arbiter** on **uhoh** which is dependent on the **sensor/danger** and on **carryon** which is dependent on the **auto produce comfy**. Locality **judgement** is expressed as a locality but each differentness of the locality closes individually with its pre arbiter source.

1.10.4. The immersed ring network

The dependency expression:

```
( ( => ) ( Z )
    Z<=behmod(initN(init2(Z)) next( )) )
```

The expressed network:

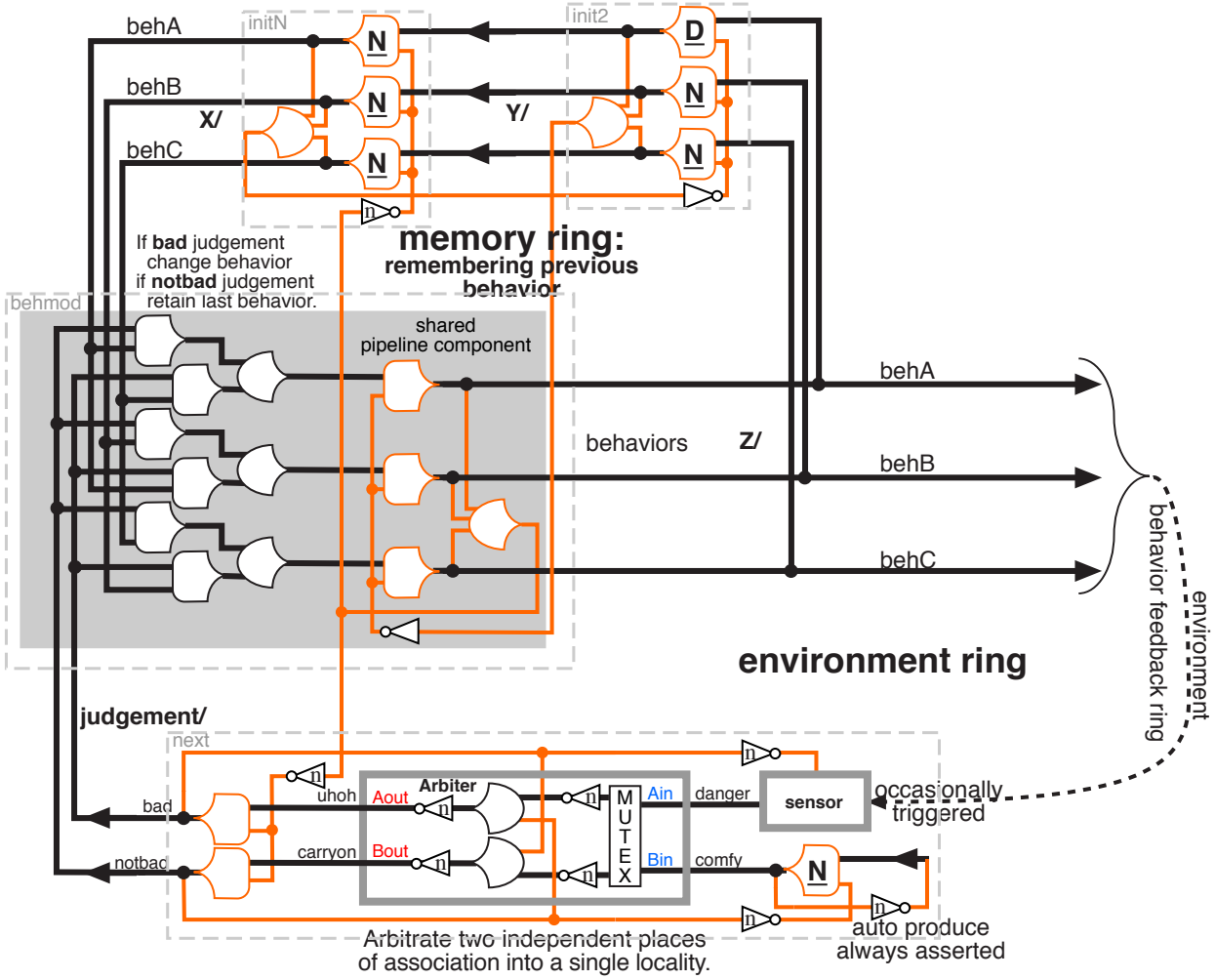


Figure 1.48. Linked rings with part of one ring flowing through the environment.

The upper ring of Figure 1.48 remembers the imposed behavior. The lower ring flowing partially through the environment determines whether the imposed behavior should change or not. If the sensor's closure is **D** the sensor will monitor the environment. If the monitoring exceeds some threshold the sensor will transition its output to **D**. When the closure becomes **N** it will transition its output to **N**. Upon which its closure will transition to **D** and the sensor will begin monitoring the environment again. A **BAD** response from the sensor might be pain or frustration or danger. In the absence of a **BAD** response from the environment the sensor's

output remains at **N** and the network remains alive from the auto produced, **NOTBAD**, continually presenting the same differentness to the environment.

The sensor **BAD** and the auto produce **NOTBAD** are happening independently and are not coordinated. They might even transition to **D** simultaneously. The arbiter receives the two uncoordinated flows **BAD** and **NOTBAD** and combines them into a single locality of mutually exclusive **judgement** conditions, presented to the network.The linked rings coupled through the interaction of **judgement** and **X** are continually alive continually imposing behavior on the environment at their intrinsic throughput rate and only occasionally does the environment respond with an indication that behavior should be changed.

If **BAD** continues to be asserted the network will continue to change its behavior until it finds a behavior that no longer asserts **BAD**. When **BAD** ceases the auto produce will play through the arbiter with **NOTBAD** and the network will continue asserting the last behavior that did not elicit **BAD**. If none of the available behaviors unassert **BAD** then the network is in either mortal danger or chronic depression.

1.10.4.1. The minimal syntax dependency expression

```
( ( => ) ( Z W X Y )
    next( => W)
    behmod( Y W => Z )
    init2(Z => X)
    (initN( X => Y ) )
```

1.10.4.2. The expanded minimal syntax dependency expression

Z inherits from both **behmod/Z** and from **init2/B** the two of which have different names for their differentnesses. But in binding portal association the syntactic structure determines and they both have the same syntactic structure. Either set of names can be inherited.

```
( ( => ) ( Z/{behA behB behC } W/{bad notbad } X/{2 1 0 } Y/{2 1 0 })
    next( => W,Z.comp)
    behmod( Y,Z.comp W,Z.comp => Z,X.comp )
    init2( Z,X.comp => X,Y.comp)
    (initN( X,Y,comp => Y,Z.comp ) )
```

While the immersed network through its own spontaneously interacting differentnesses, is manipulating and appreciating the passive differentnesses of the extrinsic environment there is nothing influencing the immersed network which is self contained, self determined and in complete control of itself.

While sequential computers are designed, implemented and manipulated by humans there is no designer and implementor of humans which must arise and behave entirely on their own expressional merits like the network above.

1.10.5. From dependence to independence

The journey began with the primitive behaviors which could be expressed with feedback or with natural hysteretic behavior such as magnetism, chemical or biomolecular such as proteins. The primitive behaviors composed into networks which themselves composed into more complex networks which were dependent on influence from an extrinsic environment for liveness, time and coordination. Closing a network with its result completeness resulted in the oscillation network (section 1.6) which autonomously expressed its own time and coordination but still relied on extrinsic influence for its liveness. Coupling oscillation networks formed pipeline networks which closing on themselves autonomously expressed their own liveness and their own sense of time independent of the environment extrinsic to the network (section 1.7.9). Closing pipeline networks into ring networks and linking ring networks formed spontaneously alive, fully autonomous complex networks possessing memory through their own appreciation of time exploring a passive environment (section 1.10).

1.11. The second level locality of interaction differentness

A second level locality extends the expressivity of locality differentnesses. A second level locality can be **"one of"** related or **"all of"** related.

1.11.1. The "one of" related second level locality

The expression:

S/{D C B A}/{2 1 0]/

Expresses a **"one of"** related second level locality named **S** containing**"one of"** related second level differentness names **D**, **C**, **B** and **A** and **"one of"** related first level differentness names **2**, **1** and **0.**

The first level locality names **/{2 1 0}/** are cross associated to the second level locality names. Each second level differentness name now contains a first level locality.

S/{ D/{2 1 0}/ C/{2 1 0}/ B/{2 1 0}/ A/{2 1 0}/ }

Dropping the dangling / and distributing the second level names to the first level names.

S/{ {D/2 D/1 D/0} {C/2 C/1 C/0} {B/2 B/1 B/0} {A/2 A/1 A/0} }

distributing the locality name.

{ S/{D/2 D/1 D/0} S/{C/2 C/1 C/0} S/{B/2 B/1 B/0} S/{A/2 A/1 A/0} }

further distributing the locality name.

{ {S/D/2 S/D/1 S/D/0} {S/C/2 S/C/1 S/C/0} {S/B/2 S/B/1 S/B/0} {S/A/2 S/A/1 S/A/0} }

The interior **"one of"** relations are redundant within the outer **"one of"** relation and are removed.

{S/D/2 S/D/1 S/D/0 S/C/2 S/C/1 S/C/0 S/B/2 S/B/1 S/B/0 S/A/2 S/A/1 S/A/0}

The result is a **"one of"** list of unique reference names. Each name a pathname to a place of association in locality **S** which has 12 places of association. **D** completeness for the locality is one place of association, asserting **D** and the rest asserting **N**. A differentness reference pathname queries whether its referenced place of association is asserting **D**.

The number of expressible mutually exclusive differentnesses for a **"one of"** related second level locality is the number of first level differentnesses times the number of second level differentnesses:

first * second

in this case 12.

1.11.1.1. Name reuse

A **"one of"** related second level locality expressivity is equivalent to a first level locality with **M*N** unique differentness names. But, if there are not **M*N** unique first level differentness names available the second level expression allows the reuse of the first level differentness names extending the use of the names. The first level differentness names are syntactically isolated within each second level differentness name so each second level differentness can reuse first level differentness names. For the locality expression:

2/{2 1 0}/{2 1 0}/

the reference

2/2/2

uniquely references one of the 9 differentnesses of the second level locality named **2**.

The expression:

1.11.2. The "all of" related second level locality

The expression:

S/[D C B A]/{2 1 0}/

Expresses an **"all of"** related second level locality named **S** containing**"all of"** related second level differentness names **D**, **C**, **B** and **A** and **"one of"** related first level differentness names **2**, **1** and **0.**

The first level locality names **/{2 1 0}/** are cross associated to the second level locality names. Each second level differentness name now contains a first level locality.

The dropping the dangling slash the above expands to:

S/[D/{2 1 0}/ C/{2 1 0}/ B/{2 1 0}/ A/{2 1 0}/]

Dropping the dangling slash and distributing the second level differentness names to the first level names.

S/[{D/2 D/1 D/0} {C/2 C/1 C/0} {B/2 B/1 B/0} {A/2 A/1 A/0}]

Forms all of the names of the 12 places of association.

Because the second level names are **"all of"** related, the interior **"one of"** relations are not redundant. A reference name of a locality differentness contains **"all of"** the second level names each associated with one first level name.

The reference:

S/[D/2 C/1 B/0 A/1]

contains 4 pathnames to places of association.

A differentnesses of the locality is expressed with a pattern of **D** assertion in contrast to a single **D** assertion. **D** completeness for the locality is four places of association, asserting **D** and the rest asserting **N**. The reference above references one mutually exclusive differentness of locality **S** with places of association **D/2** and **C/1** and **B/0** and **/A/1** asserting **D** and places of association **D/0**, **D/1** and **C/2**, **C/0** and **B/2**, **B/1** and **A/2**, **A/0** all asserting **N**. The reference to the locality differentness expresses **D** completeness for the referenced differentness of locality **S**. A differentness reference with its 4 pathnames queries whether its referenced places of association are all asserting **D**.

Because the second level differentnesses are referenced by name they can can appear in any order in a reference:

S/[C/1 A/1 B/0 /D/2]

and it still references the same locality **D** completeness configuration. However, there is another way to uniquely reference the second level locality differentnesses.

1.11.2.1. Differentness range of "all of" related second level localities

"all of" the second level names are presented in a locality reference name forming a pattern of **D** assertions in contrast to the single **D** assertion for a **"one of"** related second level locality. The**"all of"** related second level locality yields more unique differentness names than does a **"one of"** related second level locality. Each successive second level name in a second level locality differentness reference multiplies the range of locality differentness names by the number of its contained first level names. For locality **S** the number of expressible mutually exclusive differentness names is the number of first level differentness names raised to the power of the number of second level differentness names:

$$first^{second}$$

for the second level locality **S** above the range is 81 unique differentness names.

1.11.2.2. Name list ordering and differentness of place in order

The second level differentness names being expressed in a linear one dimensional expression are intrinsically ordered. Consider the ordering to be low order to high order right to left. Since all of the the second level differentness names appear in a reference to a second level locality differentness the identity of the second level differentnesses can be represented by order of appearance of their associated first level names in the reference corresponding to the order of

their appearance in the list of second level differentness names in the locality expression. For instance:

S[/D/2 C/1 B/0 A/1] /* can be expressed as */ S/[2 1 0 1]

1.11.2.3. Self bounding names

If the first level locality differentness names are single character names they are self bounding and can be presented without the bounding syntax of [ ] and interstitial bounding white space.

S/[2 1 0 1] /* can be expressed as */ S/2101

and it still references the same composite differentness name and same **D** completeness configuration.

1.11.2.4. Non self bounding names

Names that are not self bounding still require explicit syntactic bounding with white space. With the locality:

S/[D C B A]/{two one zero}/

The second level locality differentnesses can still be referenced by order but the first level names must be explicitly bounded.

S/[two one zero one]

1.11.2.5. Locality initialization

A network is by convention initialized to completely **N** (empty). There may be localities which need to be initialized to a specific **D** completeness which must be explicitly expressed. The **-** in the **T** locality expression conveniently expresses ranges from conventionally and contiguously ordered name sets such as digits and letters. The locality name **T:** is followed by the differentness name to be initialized.

T:00121100/[H-A]/{2-0}/ /* expands to */ T:00121100/[H G F E D C B A]/{2 1 0}/

Specifies that locality **T** be initialized to the differentness named **00121100**. Places of association **H/0 G/0 F/1 E/2 D/1 C/1 B/0** and **A/0** are all initialized to **D**. The remaining places of association **H/1 H/2 G/1 G/2 F/0 F/2 E/0 E/1 D/0 D/2 C/0 C/2 B/1 B/2 A/1** and **A/2** are initialized to **N**.

1.11.2.6. Generality and flexibility of locality expression

A second level locality can be arbitrarily named and structured. There does not need to be any uniformity of structure in a second level locality In particular, different first level differentness name lists can be explicitly associated with each second level differentness name The expression:

B/[W/{A R T}/ X/{B G R}/ Y/{M N}/ Z/{1 0}/]

expresses a second level locality named **B** containing four **"all of"** related second level differentnesses named **W**, **X**, **Y** and **Z**. Second level name **W** contains first level differentness names **A**, **R** and **T** representing three places of association. Second level name **X** contains first level differentness names **B**, **G** and **R**. representing three places of association. Second level name **Y** contains first level differentness names **M** and **N**. representing two places of association. Second level name **Z** contains first level differentness names **1** and **0**. representing two places of association. The locality expresses **3x3x2x2 = 36** uniquely named differentnesses each representing a unique pattern of **D** completeness of the locality.

references are

B/[W/T X/R Y/N Z/0] /* References one differentness of locality B */
/* relying on list order and the self bounding of names */
B/TRN0 /* References the same differentness name as above */
B/W/T /* References place T of locality W of locality B */
B/X /* References locality X of locality B */
B /* References locality B as a whole */

1.11.2.7. Nature's second level localities

Nature constructs large ranges of mutually exclusive differentness with second level differentness localities composed of smaller first level localities of differentness.

1.11.2.7.1. DNA

DNA is an association of four **"one of"** related conditions **"all of"** associated three at a time (codon).

Gene/[?:0]/[2 1 0]/{A C G T} /* expresses a gene with an unspecified number of codons */

Notice there is no dangling slash /**{A C G T}** are names of differentness conditions primitive to the expression of the gene. They correspond to the above /**{D N}**. Also notice that there is no **"not a differentness"** condition because the conditions are not transitioning. A gene does not actively interact but is passively interpreted and it is the responsibility of the interpreter to delimit the differentnesses. The names **2 1 0** are the differentness names of the first level differentness of the locality. A gene is one expressed differentness of an **"all of"** related second level locality.

DNA is reflexive. Capable of expressing proteins and enzymes that manipulate and manage the DNA itself.

1.11.2.7.2. Protein

A protein polypeptide is an ordered association of 20 **"one of"** related amino acid conditions.

protein/[?:0]/{Ala Arg Asn Asp Cys Gin Glu Gly His Lie
Leu Lys Met Phe Pro Ser Thr Trp Tyr Va}
/* expresses a protein with an unspecified number of amino acids */

The folding conformation of the protein may or may not be characterizable in the language in terms of dependency relations. Above my pay grade. Again there is no dangling /. The twenty amino acid abbreviations are names of differentness conditions primitive to the expression. Again there is no **“not a differentness”** condition because the protein acts by changing its folding configuration not by transitioning its component differentnesses. A protein is one expressed differentness of an **“all of”** related first level locality of differentness.

1.11.3. Forming second level locality differentness names

Two interacting first level localities **A/{2 1 0/}/** and **B/{2 1 0/}/,** see section 1.4.6, cross associate their names forming composite names expressing a larger range of mutually exclusive differentness each of which is recognized and mapped to a specific result differentness. This suggests that two first level localities might cross associate simply to form a larger range of mutually exclusive differentnesses with each differentness uniquely named with a composite differentness name formed of first level differentness names (see Figure 1.12). Then a third first level locality might cross associate with those composite differentness names to form a larger range of mutually exclusive differentness with larger composite differentness names, and so on. This progressive cross association is how a second level locality expresses a larger range of mutually exclusive differentness names.

1.11.3.1. Intrinsic order

It is already established above in section 1.11.2.2 that the second level differentness name with self bounding symbols can rely on the order of the second level differentness names in the locality expression to express the second level differentness name. Using the order of the first level differentness names and the order of the second level differentness names in the locality expression to determine the ordering of the progressive cross association results in a specific ordering of the locality differentness names.

1.11.3.2. Ordered progressive cross association

The **“all of”** related second level locality

W/[C B A]/{2 1 0}/

expands to

W/[C/{2 1 0}/ B/{2 1 0}/ A/{2 1 0}/]

The first level differentness names associated with lowest order second level differentness name **A/{2 1 0}** is cross associated by associating all the first level differentnesses of **A** in order with each first level differentness name of **B/{2 1 0}** in order forming an ordered list of mutually exclusive composite (**BA**) differentness names.

22 21 20 12 11 10 02 01 00

The ordered differentness names are then cross associated low order to high order with the first level differentness names associated with the next higher order second level differentness **C/{2 1 0}** appending its first level differentness names to the high order end of the growing

composite name forming a larger range of ordered mutually exclusive composite (**CBA**) differentness names

022 021 020 012 011 010 002 001 000
122 121 120 112 111 110 102 101 100
222 221 220 212 211 210 202 201 200

The range of ordered differentness names is then cross associated low order to high order with the first level differentness names associated with each next higher order second level differentness, and so on

This progressive cross association is illustrated in Figure 1.49. First level differentness names are low order to high order right to left and second level differentness names are low order to high order bottom to top along the right edge. The **"all of"** related second level locality composite differentness names low order to high order right to left are along the bottom of each figure.

Three first level differentnesses are used to make it different from the familiar binary but not so unfamiliar as to be confusing.

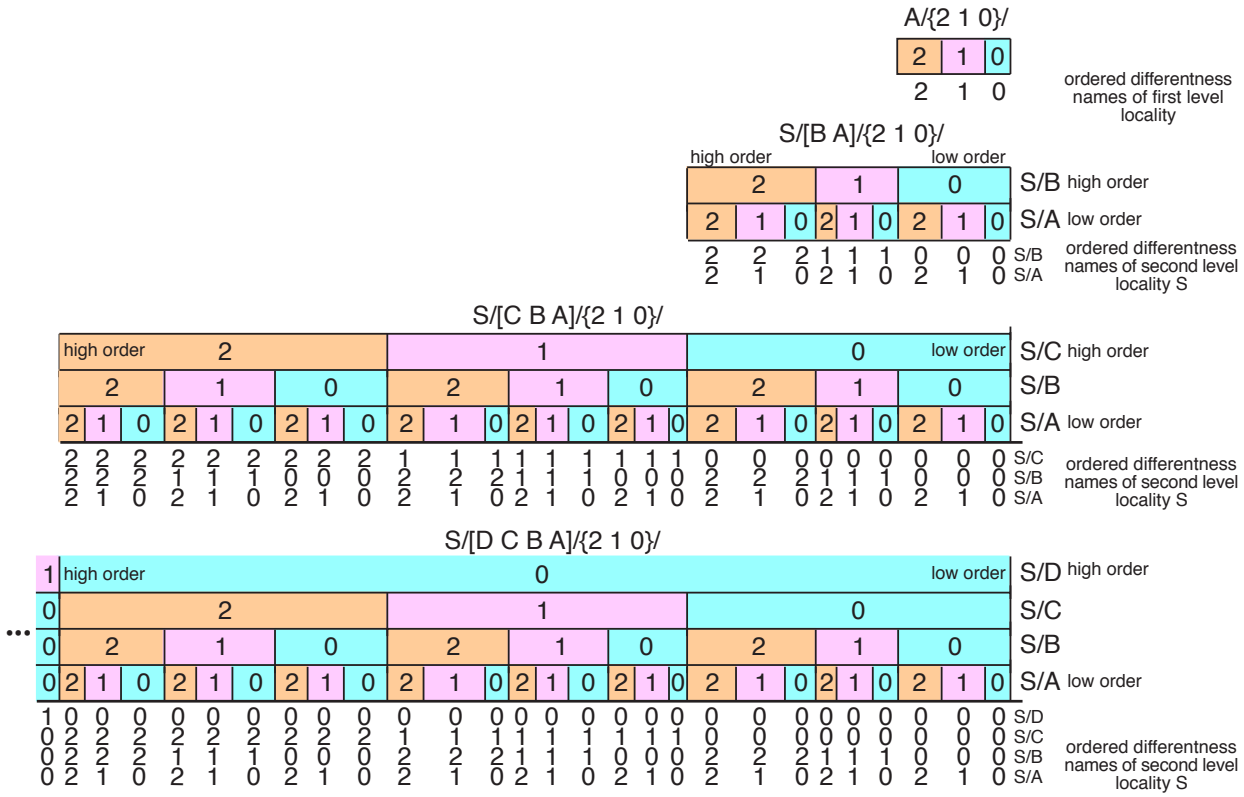


Figure 1.49. Generation of the ordered name list with successive cross associations.

The name list is a discrete list of mutually exclusive differentness names formed and ordered in a particular methodical way in relation to the locality expression. There is nothing between neighboring names. There is no implication of uniformity of interval or of continuity of range

associated with the names. The graphical name ranges are purposely drawn non uniformly to emphasize this. It is just a list of names.

1.11.4. An emergent differentness

The progressively ordered cross association produces a specifically ordered list of unique composite differentness names. In addition to each differentness name being uniquely different from all the other differentness names each name now represents a uniquely different place of order in the list of composite differentness names ordered low order to high order.

Every ordered name list begins with a lowest order name. Every differentness name has a unique place of order in the name list relative to the lowest order name. Place in order of second level differentness names can be compared across differently expressed second level localities with one to one correspondence of their respective name lists from their lowest order names. Figure 1.50 illustrates this one to one correspondence from lowest order names with the name lists of four second level localities with the names of the corresponding place order highlighted.

An “all of” related second level locality differentness name now represents two forms of differentness, absolute differentness and differentness of order in name list.

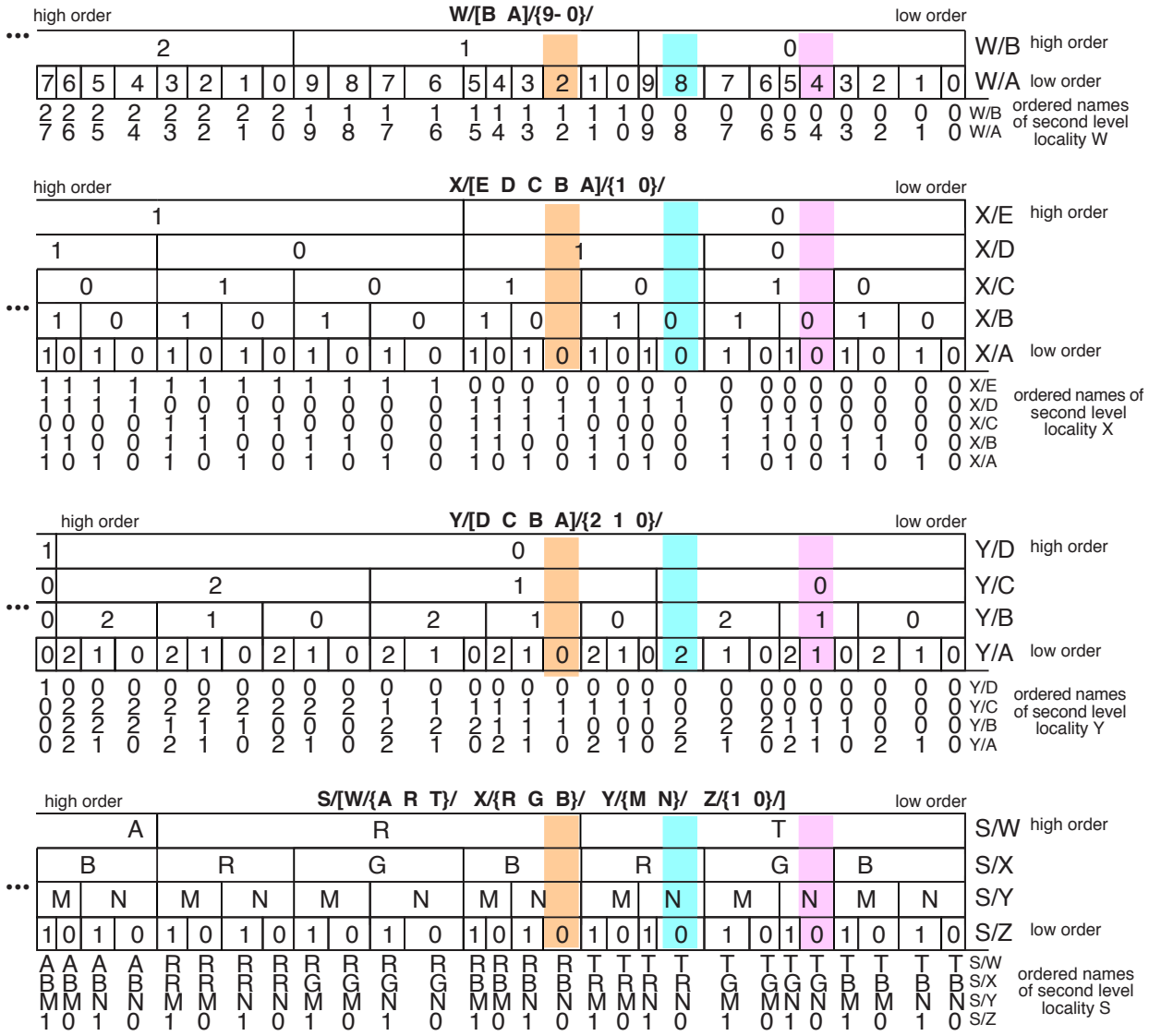


Figure 1.50. One to one correspondence of different name lists.

Names with equal place of order from the above name lists.

W/12 = X/01100 = Y/0110 = S/RBN0

W/08 = X/01000 = Y/0022 = S/TRN0

W/04 = X/00100 = Y/0011 = S/TGN0

There can be a standard representation of range of differentness into which heterogeneous ranges of differentness can be converted for comparison and interaction.

1.12. Interacting second level locality differentnesses

To interact two second level localities one could directly cross associate the entire ranges of differentness and map each cross association to a specific differentness. But the large ranges of differentness of second level localities makes the full cross association impractical. Perhaps a

more practical approach to second level locality differentness interaction might be found in terms of the newly emerged differentness of place of order in the list of second level locality differentness names.

After making a point of the one to one correspondence between abstract relations among abstract differentnesses and their realizable network D completeness relations among real association related differentnesses it may seem odd to be talking about abstract relations among abstract names of an abstract ordered list of names with no direct correspondence to realizability. Stay tuned. By the end of this section the purely abstract will map to the directly realizable.

1.12.1. Interacting place in order

The unique place in order of a composite differentness name in the ordered name list of a second level locality's differentnesses is characterized as the range of name boundaries crossed between the composite differentness name and the lowest order name of the name list. If a differentness name corresponds to the lowest order name of a name list no boundaries are crossed.

Figure 1.51 illustrates the interactive combination of boundary crossings for differentness name **M/0010** and differentness name **N/0011**. The **M/0010** name range extends three boundary crossings from the lowest order name **0000** of the name list illustrated by the top **green** line on the diagram. The **N/0011** name range extends four boundary crossings from the lowest order name **0000** of the name list illustrated by the top **red** line on the diagram. The differentnesses interact by combining their boundary crossings. Begin with the boundary crossings of the differentness name **M/0010** range from the lowest order name **0000** of the name list. Extend the boundary crossings of **M/0010** by the four boundary crossings of **N/0011** with one to one correspondence along the name list. The list name **X/0021** that the last boundary crossing falls into is the differentness name range representing the result of the interactive combination of differentness named **M/0010** and differentness named **N/0011**.

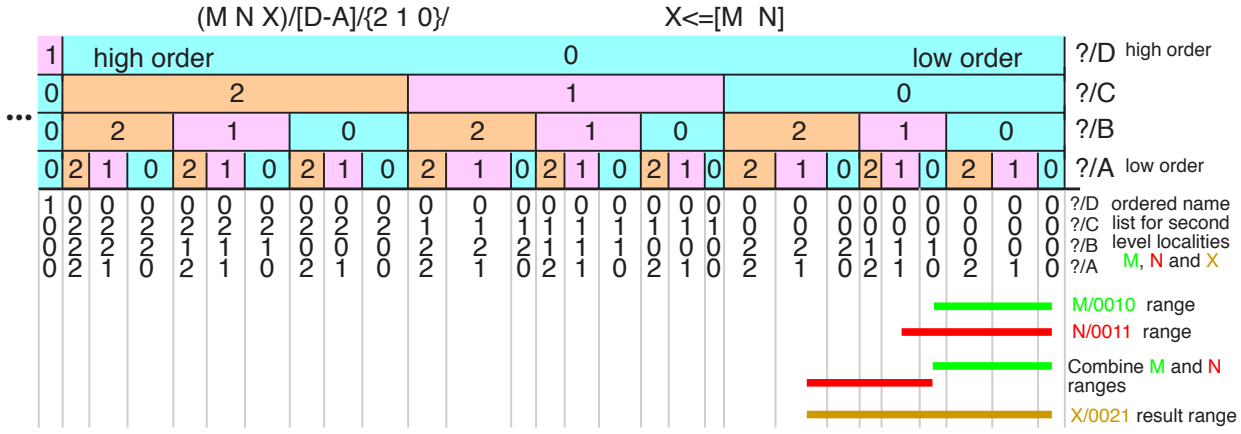


Figure 1.51. Combining names in relation to places of order in the name list.

1.12.2. Second level names in terms of their first level names

A second level differentness name place in order can be represented in terms of the combination of its first level network locality name places in order. Each first level differentness

name has a place in order on the name list in relation to its second level differentness names. The upper portion of Figure 1.52 shows the first level name places in order in relation to their associated second level differentness names. The lower portion shows the place in order of second level locality differentness names in terms of the combination of their first level name places in order.

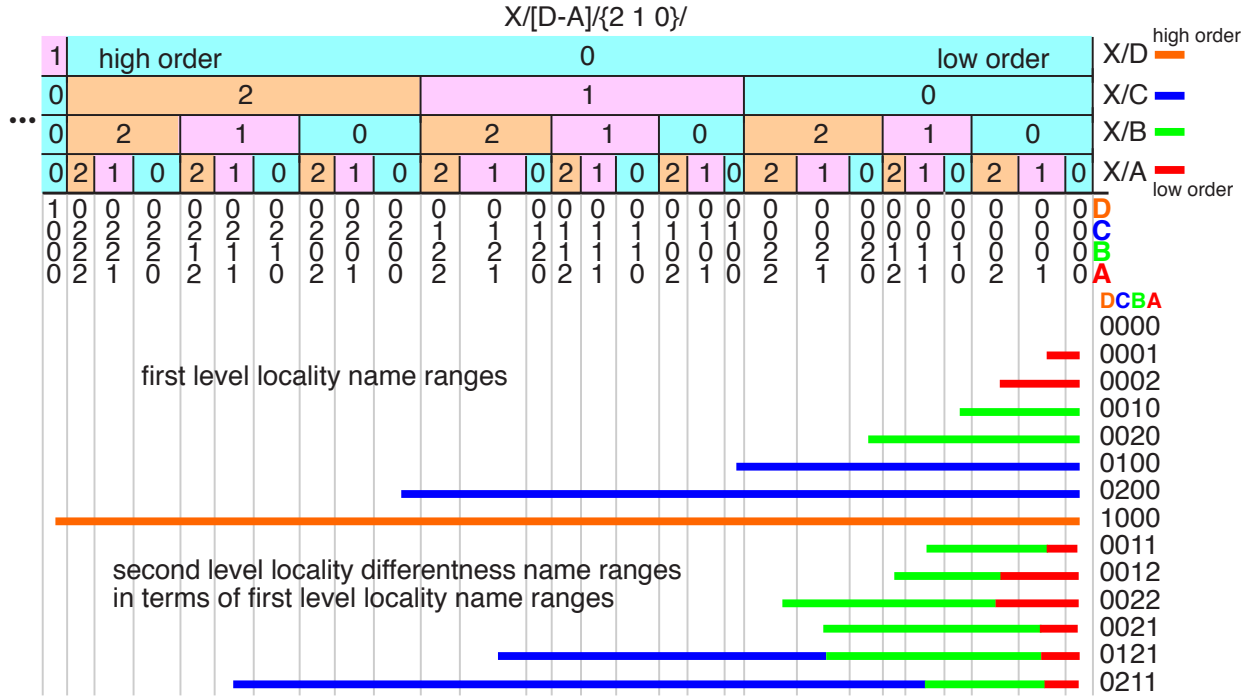


Figure 1.52. Second level name ranges in terms of first level name reanges.

1.12.3. Combining second level locality names in terms of their first level locality name ranges

It follows that second level locality differentness names can interact by accumulating the combinations of their first level name places in order. Figure 1.53 illustrates the combination of second level name places in order in terms of combining and accumulating their corresponding first level name places in order. Each result name is then represented in terms of its intrinsic first level places in order.

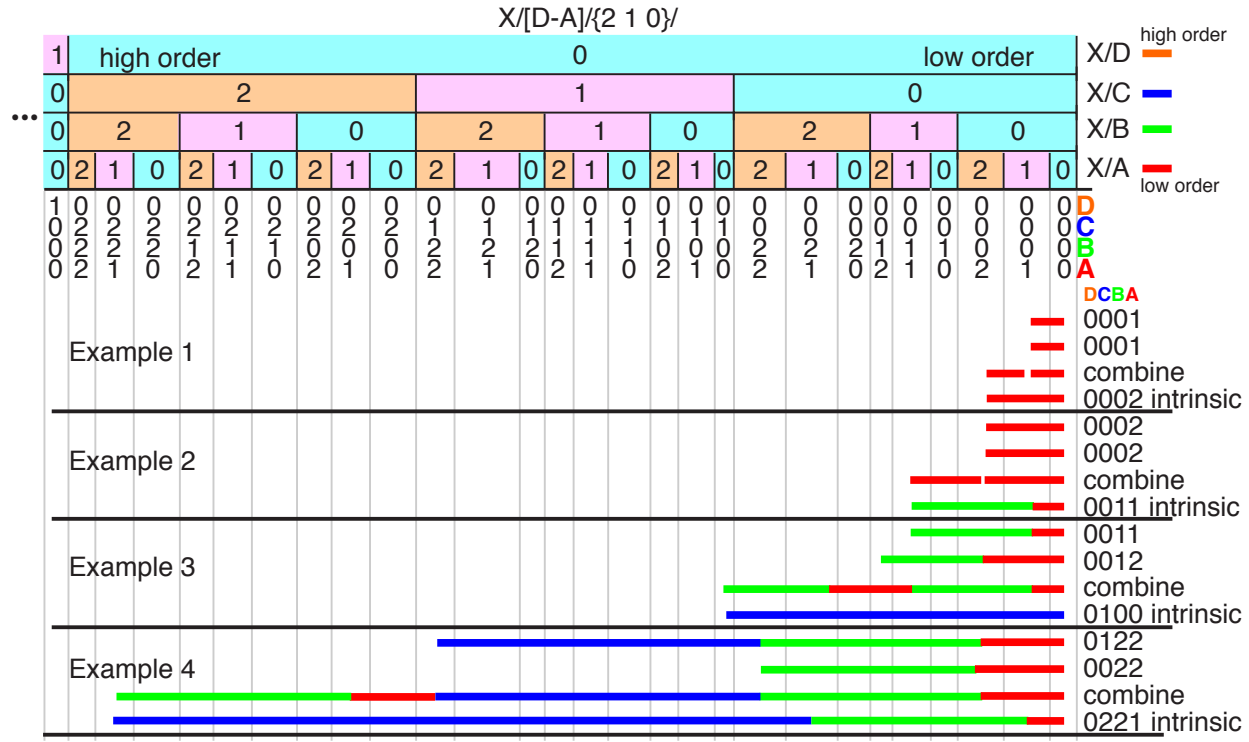


Figure 1.53. Combining second level differentness names in terms of their first level differentness name places in order.

1.12.4. The common structure and proportionality

The first level differentness names associated with corresponding second level differentness names are identically structured and have proportional ranges of place relation to their second level differentness names. Then first level differentness names associated with interacting corresponding second level differentnesses can be combined with a common methodology.

1.12.5. Interacting first level differentness names

Second level differentness names can interact with progressive combination of first level differentness names associated to corresponding second level differentness names from the lowest order second level differentness names to the highest order second level differentness names.

When fist level differentness name ranges are combined the result ranges will extend beyond the highest order first level differentness name range. This is accommodated by passing the excess range of combination to the next higher second level differentness name to influence its combination.

Given second level localities **X**, **Y** and **Z**

X/[D C B A]/{2 1 0}/ Y/[D C B A]/{2 1 0}/ Z/[D C B A]/{2 1 0}/

The following examples combine localities **X** and **Y** to locality **Z**.

Z<=[X Y]

1.12.5.1. Combining first level differentness names of the lowest order second level differentness name

Figure 1.54 shows the combination of the lowest order second level differentness names in terms of their first level differentness name name list ranges. The combination of lowest order second level names does not receive influence from a lower order combination of second level names.

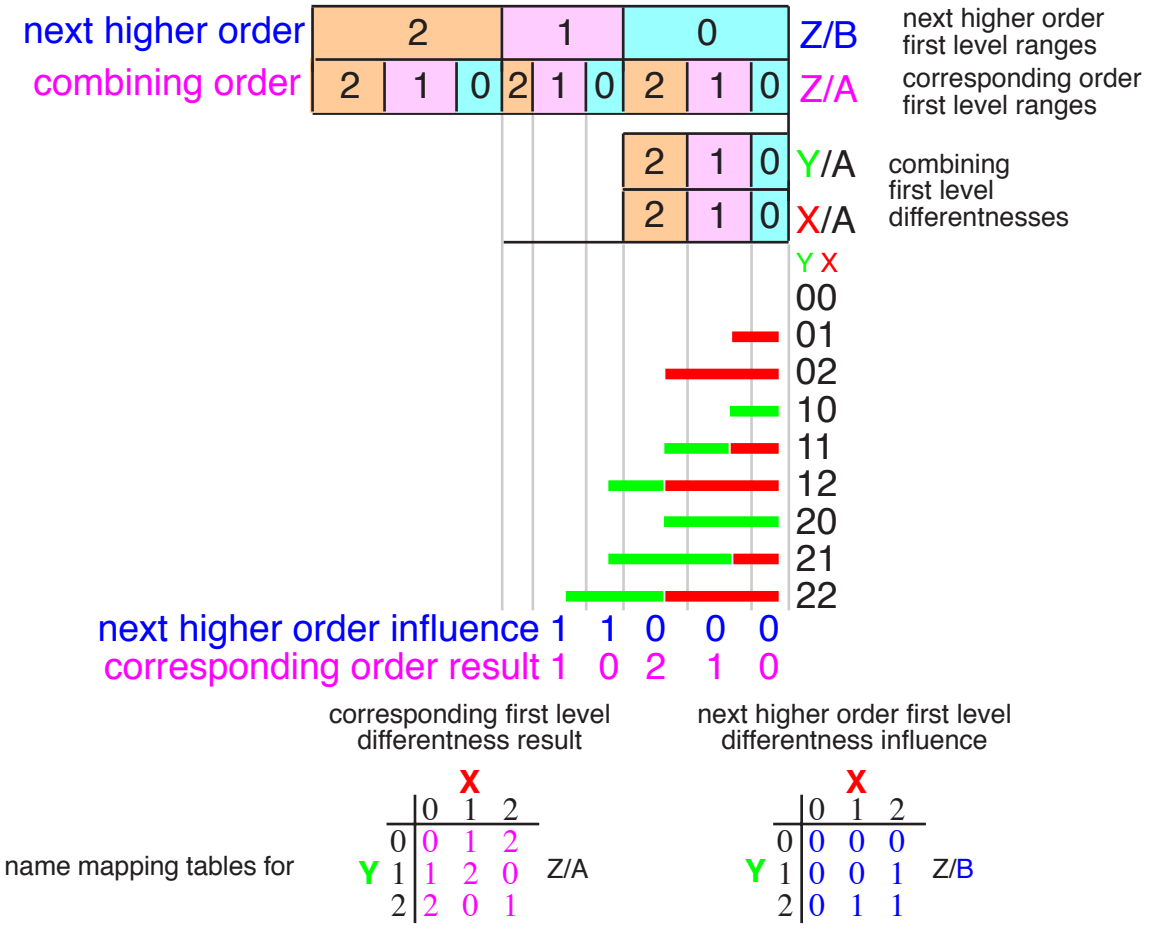


Figure 1.54. First level name combination of corresponding lowest order second level differentness names.

The diagram of Figure 1.54 represents the cross association of interacting first level differentness names with each cross association result name determined in terms of range in relation to the name list of the composite cross association name. Each cross association of first level differentness names has a specific result differentness name so the combination can be expressed as a mapping of first level differentness names with the mapping tables at the bottom of Figure 1.54. The table at the left is the cross association mapping for the first level differentness names of the lowest order second level differentness name. The table at the right is

the cross association mapping for the influence name to the next higher second level differentness name combination.

1.12.5.2. Combining higher order second level differentness names

Each higher order second level differentness combination, shown in Figure 1.55, includes the influence first level name range from its next lower order second level differentness name combination as well the first level name ranges associated with its combining **X** and **Y** second level differentnesses.

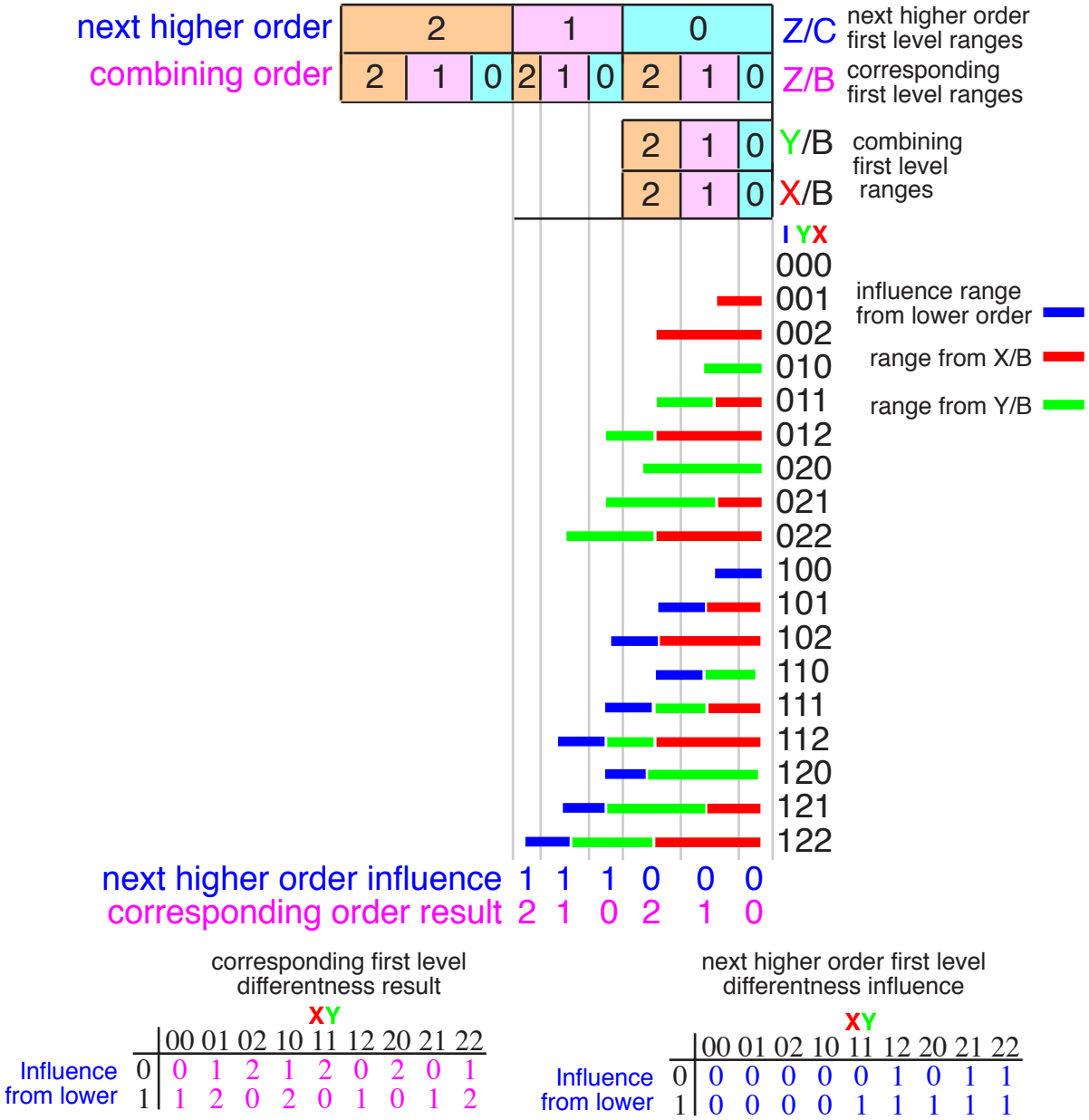


Figure 1.55. Higher order first level locality combination with influence range.

The combination diagram of Figure 1.55 maps each cross association of first level differentness names with name ranges proportional to the second level differentness name to which they are associated to the result first level name and to the next higher level influence first level name. The direct mapping of names is represented in the tables at the bottom of Figure 1.55.

1.12.5.3. Progressive combination of first level localities

Because each higher order second level differentness name combination must wait to receive the influence from its next lower order second level differentness name combination the dependency relations for the combination of second level locality differentness names progress in order from the lowest order second level differentness name to the highest order second level differentness name. When the combination of the highest order second level differentness name is completed the corresponding second level locality combination result differentness name is fully formed and the place in order range of the second level locality combination result name has been fully accumulated.

1.12.5.4. Dependency expressions derived from the tables

The combination of second level locality differentnesses is now characterized entirely in terms of cross association and mapping relations among first level differentness names. It no longer refers to name range or order in relation to the ordered name list.

The dependency language specifications for the first level locality combinations are read directly from the name mapping tables.

```
(lowestorder(A/{2 1 0}/ B /{2 1 0}/ => Z/{2 1 0}/ influenceout/{1 0}/ )
        /* halfadder */
/* dependency relations from lowest order result name mapping table of Figure 1.54 */
Z/{0<={[A/1 B/2] [A/2 B/1] [A/0 B/0]}
   1<={[A/1 B/0] [A/0 B/1] [A/2 B/2]}
   2<={[A/2 B/0] [A/0 B/2] [A/1/ B/1]} }
/* dependency relations from the lowest order influence name table of Figure 1.54 */
influenceout/{1<=[A/2 B/1] [A/1/ B/2] [A/2 B/2]
              0<={[A/0 B/0] [A/1 B/0] [A/0/ B/1] [A/1/ B/1] [A/0/ B/2]} [A/2/ B/0] } )
```



```
(higherorder(A/{2 1 0}/ B/{2 1 0}/ Influencein/{1 0}/ => Z/{2 1 0}/ influenceout/{1 0}/ )
        /* fulladder */
/* dependency relations from the higher order resultname table */
Z/{0<={ [A/0 B/0 Influencein/0] [A/1 B/2 Influencein/0] [A/2 B/1 Influencein/0]
        [A/0 B/2 Influencein/1] [A/2 B/0 Influencein/1] [A/1 B/1 Influencein/1] }
   1<={ [A/0 B/1 Influencein/0] [A/1 B/0 Influencein/0] [A/2 B/2 Influencein/0]
        [A/0 B/0 Influencein/1] [A/1 B/2 Influencein/1] [A/2 B/1 Influencein/1] }
   2<={ [A/2 B/0 Influencein/0] [A/0 B/2 Influencein/0] [A/1 B/1 Influencein/0]
        [A/1 B/0 Influencein/1] [A/0 B/1 Influencein/1] [A/2 B/2 Influencein/1] }
 }
/ dependency relations from the higher order influence name table
influenceout/{0<={[A/0 B/0 Influencein/0] [A/1 B/0 Influencein/0] [A/0 B/1 Influencein/0]
                  [A/2 B/0 Influencein/0] [A/0 B/2 Influencein/0] [A/1 B/1 Influencein/0]
                  [A/0 B/0 Influencein/1] [A/1 B/0 Influencein/1] [A/0 B/1 Influencein/1]}
               1<={[A/1 B/2 Influencein/0] [A/2 B/1 Influencein/0] [A/2 B/2 Influencein/0]
                  [A/0 B/2 Influencein/1] [A/2 B/0 Influencein/0] [A/1 B/1 Influencein/1]
                  [A/1 B/2 Influencein/1] [A/2 B/1 Influencein/1] [A/2 B/2 Influencein/1]}
               }  )
```

Below is the expression for interacting second level localities **A** and **B** each with five second level differentnesses producing a second level result locality **S** with six second level differentnesses.

```
(combinesecond(A/[4-0]/{2 1 0}/ B/[4-0]/{2 1 0}/ => S/[5-0]/{2 1 0}/)
              ( infl/[5-1]/{1 0}/ )
  lowestorder(A/0 B/0 => S/0 infl/1)
  higherorder(A/1-4 B/1-4 infl/1-4) => S/1-4 infl/2-5 )
  S/5/{0<=infl/5/0
       1<=infl/5/1 }    )
```

expands to

```
(combinesecond(A/[4-0]/{2 1 0}/ B/[4-0]/{2 1 0}/ => S/[5-0]/{2 1 0}/)
              ( infl/[5-1]/{1 0}/ )
lowestorder(A/0 B/0 => S/0 infl/1)
higherorder(A/1 B/1 infl/1 => S/1 infl/2 )
higherorder(A/2 B/2 infl/2 => S/2 infl/3 )
higherorder(A/3 B/3 infl/3 => S/3 infl/4 )
higherorder(A/4 B/4 infl/4 => S/4 infl/5 )
  S/5/{0<=infl/5/0
       1<=infl/5/1 }    )
```

Figure 1.56 shows the **lowestorder** and the **higherorder** combination networks and the **combinesecond** network which combines two second level localities each with 5 second level differentnesses.

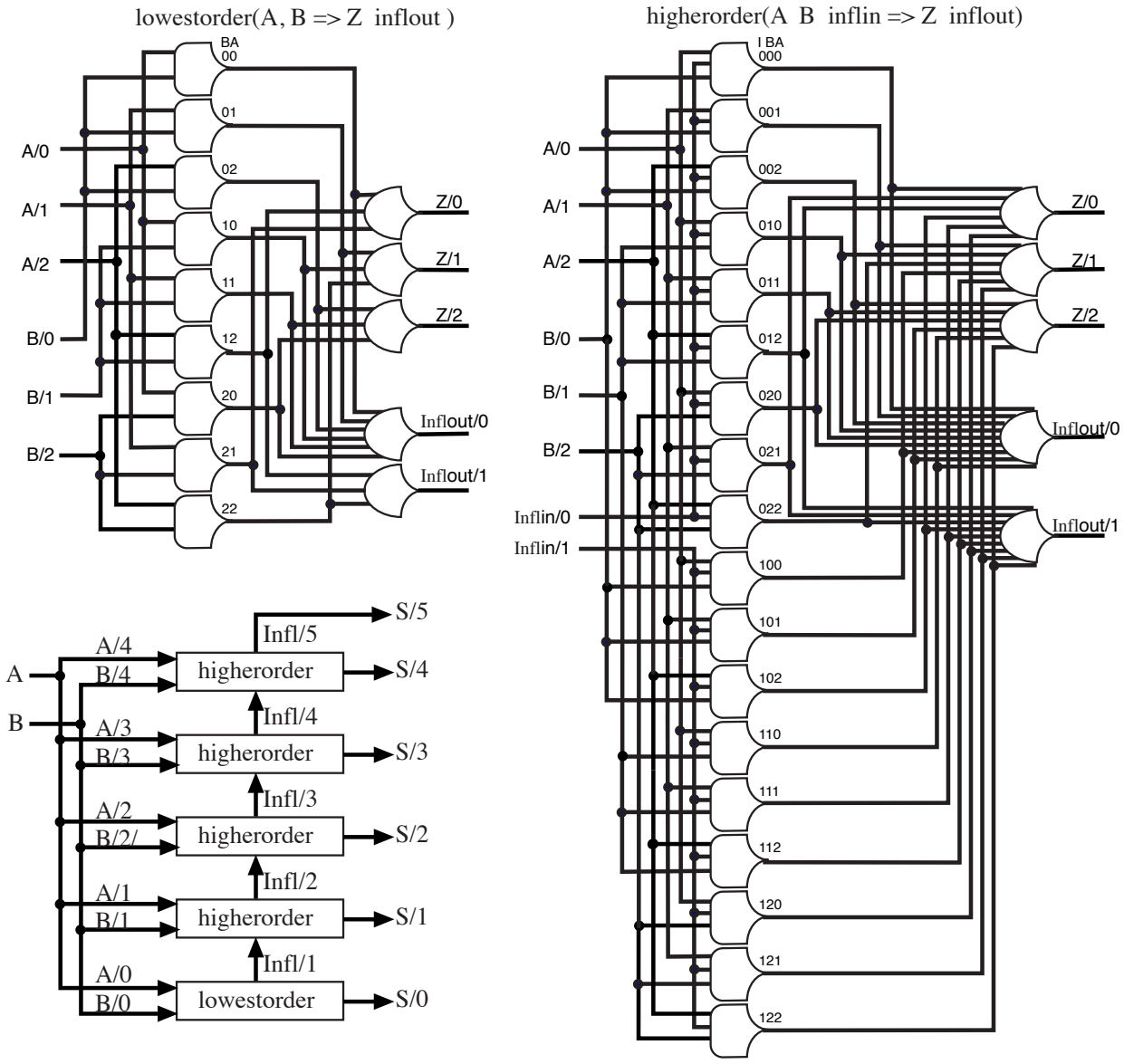


Figure 1.56. The networks generated from the expression.

1.12.6. Place value numeral and its arithmetic

A second level locality differentness name with its ordered second level differentness names each associated with a first level differentness name corresponds to the places and values of place value numerals. The ordered list of second level locality differentness names is generated with progressive cross association ordered by the order relations of differentness names in the expression of the locality. There is no appeal to counting. There is no appeal to the notion of successor. There is no successor to the highest order differentness name of a second level locality. There is no appeal to the notion of zero. The lowest order relation intrinsically fulfilled

the role of zero which did not have to be invented. There is no assumption about uniformity of intervals between names or of continuity of differentness from name to name.

We did not have to invent place value numerals. They are latent in and emerge from the dependency language. Each second level locality differentness name is a place value numeral but it is not a mathematical number. It is a computation numeral. The list of differentness names is not a number line. It is just an ordered list of differentness names.

The derivation of second order locality differentness name interaction in terms of proportional accumulation of their first order differentness name ranges in relation to the name list maps directly to the place value numeral addition algorithm in terms of digits.

Place value numerals and their arithmetic have been derived from linguistic syntax rather than from counting and the number line.

1.12.7. A question

There is a fundamental differentness between number and numeral. A numeral is generally thought of as an abstract representation of a real number.

> Addition, however, is not *au fond* and operation on representations; strictly speaking, it is defined only over numbers. The idea of "eight-bit number" thus relies on a derivative extension of bits to number and of addition to numerals, via an assumed numeral-number representation relation.
>
> Brain Cantwell Smith*

But, what if there is no assumed numeral-number relation? We just derived numeral with no reference to number. And, what if addition is *au fond* an operation on representation. We just derived addition in terms of the representation of numeral.

1.12.7.1. What is abstract? What is concrete?

The question becomes which deserves precedence of significance? Which is "more real"? Is numeral an abstraction of number or is number an abstraction of numeral?

Nature invented numeral representation long before humans invented the notion of number. DNA, RNA and polypeptides are place value numeral representations.While nature has not used addition or order comparison in relation to its numerals it uses them to amplify the expression of differentness, to express a very large range of unique differentnesses; in particular the differentnesses of proteins.

If number is derived from counting the act of counting is, in form, closer to numeral than to number. It is discrete. It does not go on forever. When the count is done it's the end. It does not involve zero. It is real

The notion of number, uniformity, continuity, order, succession, the number line, can all be extrapolated from the expression of numeral.

In the context of the language a second order locality, a numeral, is not just representational it is expressional. It can be realized directly and concretely, not just by interpretation. In other

*. Smith, Brian Cantwell, *Computational Reflections*. Cambridge, MA: MIT Press, 2026. p 22

words numeral is intrinsically concrete while number hovers as an abstract tool, as a Platonic form, realized as numeral.

1.12.8. Appreciating second level locality transition completeness

Appreciating the transition of the result locality to **D** completeness and to completely **N** is the first step towards expressing self regulation. Each reference to a locality differentness represents a **D** completeness configuration of the locality.

Completeness of transition of a second level locality is appreciated in terms of transformation of the expression of the locality into an effective dependency expression. The locality expression is transformed with an ordered cross association of differentness names that forms all of the possible references to the locality within the primitive association relations.

A/[1 0]/{1 0}/

The locality expression is being transformed into references to the locality. Locality referenc names do not include the {**D N**} term so the dangling slash is dropped.

A/[1 0]/{1 0}

The first level differentness names **/{1 0}** are distributed across the second level differentness names **[1 0]**

A/[1/{1 0} 0/{1 0}]

Then the second level differentness names **1/** and **0/** are distributed across their associated first level differentness names forming first level locality references.

A/[{1/1 1/0} {0/1 0/0}]

Then distribute locality name **A/** to form second level locality pathname references within the association relations forming a dependency expression of **D** completeness for the locality.

[{A/1/1 A/1/0} {A/0/1 A/0/0}]

Express a locality with a single place of association

A.comp/

and make it dependent on the completeness expression with locality nesting

A.comp<=[A/{1/1 A/1/0 } {A/0/1 A/0/0}]

The resulting expression maps to network **two bit** of Figure 1.57. Locality **A** monotonically transition between **D** completeness a completely **N**. **D** completeness for locality **A** is **“one of”** "**A/1/1** or **A/1/0** and **“one of” A/0/1** or **A/0/0** transitioned to **D**. The derived expression transitions to **D** when the locality **A** transitions to **D** completeness and transition to **N** when locality **A** transitions to completely **N**. The transition of **A.comp** to **D** represents the transition to

D completeness of the locality. The transition of **A.comp** to **N** represents the transition to completely **N**, emptiness, of the locality.

The expansion of a locality to its completeness expression is represented with a **?**. The above expression contracts to.

```
( A/[1 0]/{1 0}   A.comp/ )
A.comp<=?A
```

Locality expression expansion for the **five bit** completeness network of Figure 1.57.

```
B/[4-0]/{1  0}
B/[4  3  2  1  0]/{1  0}
B/[4/{1  0} 3/{1  0}  2/{1  0}  1/{1 0}  0/{1  0}]
B/[{4/1  4/0}  {3/1  3/0}  {2/1  2/0}  {1/1  1/0}  {0/1  0/0}]
[{B/4/1  B/4/0}  {B/3/1  B/3/0}  {B/2/1  B/2/0}  {B/1/1  B/1/0}  {B/0/1  B/0/0}]
B.comp<=[{B/4/1  B/4/1}  {B/3/1  B/3/0}  {B/2/1  B/2/0}  {B/1/1  B/1/0}  {B/0/1  B/0/0}]
B.comp<=?B
```

Locality expression expansion for the **five trit** completeness network of Figure 1.57.

```
C/[4  0]/{2 1 0}/
C/[4-0]/{2 1 0}
C/[4  3  2  1  0]/{2 1 0}
C/[4/{2 1 0} 3/{2 1 0}  2/{2 1 0}  1/{2 1 0}  0/{2 1 0}]
C/[{4/2  4/1  4/0}  {3/2  3/1  3/0}  {2/2  2/1  2/0}  {1/2  1/1  1/0}  {0/2  0/1  0/0}]
[{C/4/2  C/4/1  C/4/0}  {C/3/2  C/3/1  C/3/0}  {C/2/2  C/2/1  C/2/0}
                        {C/1/2  C/1/1  C/1/0}  {C/0/2  C/0/1  C/0/0}]
C.comp<=[{C/4/2  C/4/1  C/4/0}  {C/3/2  C/3/1  C/3/0}  {C/2/2  C/2/1  C/2/0}
                        {C/1/2  C/1/1  C/1/0}  {C/0/2  C/0/1  C/0/0}]
C.comp<=?C
```

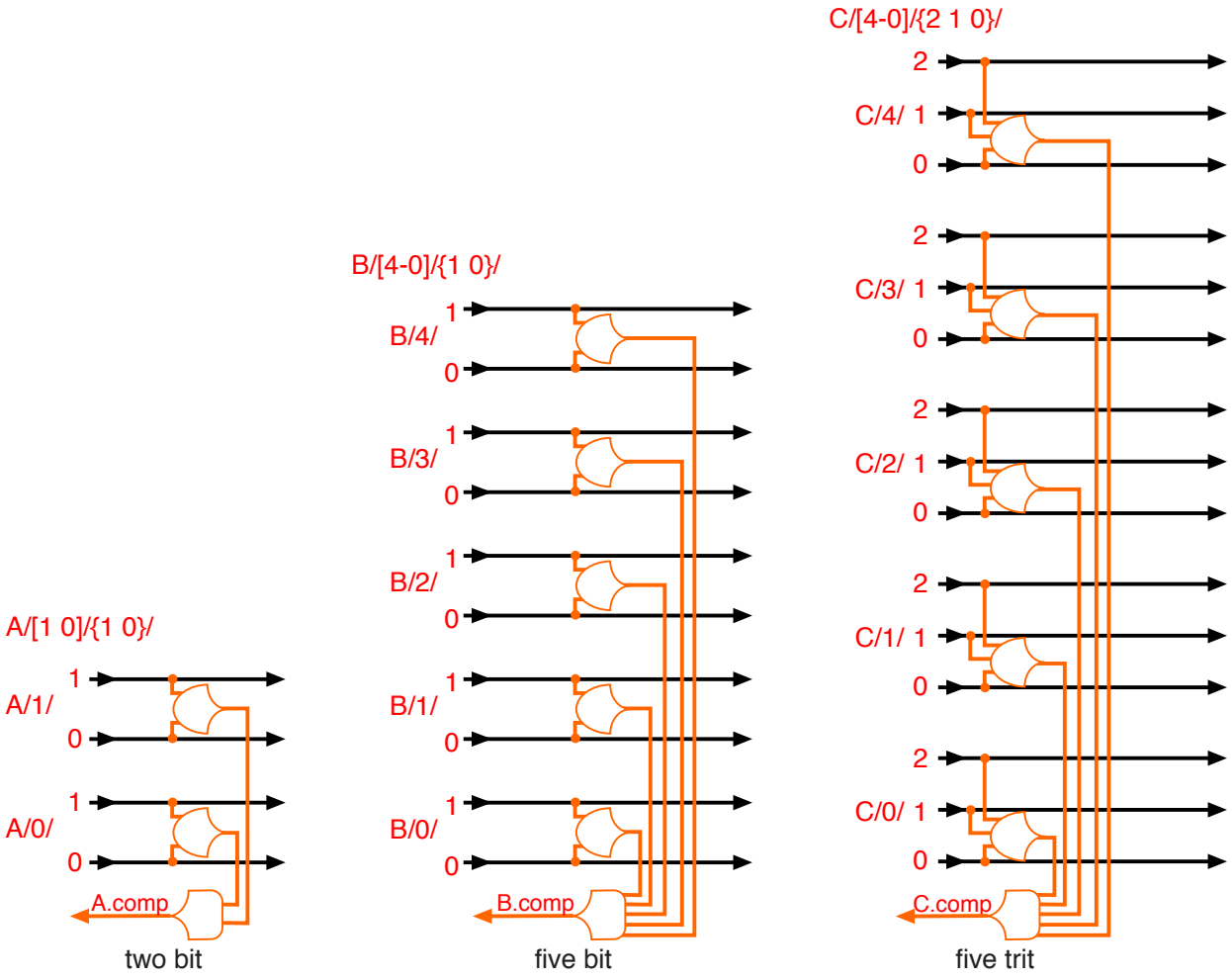



Figure 1.57. Completeness networks derived from their locality expressions.

Each network reduces the transitions to completeness of its locality to transitions of a single place of association locality with suffix **.comp**. Each **x.comp** is the singular appreciation of the singularly appreciable transitions to completeness of each second level locality.

1.12.9. The five bit add as a half oscillation

```
(fivebitadd (A/[4-0]/{2 1 0}/,AB.comp/ B/[4-0]/{2 1 0}/,AB.comp/ => S/[5-0]/{2 1 0}/,S.comp/)
            ( infl/[5-1]/{1 0}/ )
AB.comp<=?[S{/0<=lowestorder(A/0 B/0 => # infl/1)
             1<=higherorder(A/1 B/1 infl/1 => # infl/2 )
             2<=higherorder A/2 B/2 infl/2 => # infl/3 )
             3<=higherorder A/3 B/3 infl/3 => # infl/4 )
             4<=higherorder(A/4 B/4 infl/4 => # infl/5 )
             5<=infl/5
              S.comp] )
```


lowestorder and **higherorder** are the first expressions encountered that have binding portals with two result localities. A nesting relation is one to one and a binding portal reference can only support one resting relationship. When there are two or more result localities the # indicates that the result locality of this syntactic location in the binding portal that is associated with a nesting relation while the others are associated with name correspondence.

---

1. Possibly because syntax relations exhibit intrinsic structure and name correspondence relations do not there has been a concerted effort in mathematics to eliminate or at least minimize the use of names with combinators, lambda calculus, FP and so on. But there will always be a bit of name correspondence in any expression of dependency relations. Syntax structure and name correspondence, similar to electromagnetism, each regenerate the other to indefinite extension. The sameness of syntax engenders name differentness. The sameness of name engenders syntax differentness.
2. See Journey Through Computation Chapter 3
3. The **mutex/arbiter** is a well understood component of asynchronous design.
   Teresa Meng, "Synchronization design for digital systems", (Boston, Kluwer Academic Publishers, 1991) pp 158-163
4. The clock phases indicate empty and full.
   The monotonic transitioning convention is also consistent with conventional computation. Conventionally a computation is invoked, instantiates empty, input is presented, it delivers an output and then uninstantiates, disappears, emptying the computation. That each successive computation is a new instantiation of an empty computation that performs and disappears is a form of decoupled monotonicity. A flow computation does not uninstantiate and disappear but stays instantiated and is explicitly emptied after each computation representing a form of coupled monotonicity. Present input, empty, present input, empty, present input, empty ... and so on.
5. The closures linking networks is similar to the notion of asynchronous handshakes.

   > "in its most general form, asynchronous design removes the global clock in favor of distributed local handshaking to control data transfer and changes of state."
   >
   > Peter A. Beerel, Recep O. Ozdag, Marcos Ferretti, "A Designers Guide to Asynchronous Design", (New York, Cambridge University Press, 2010) p. 1

   This is a view that defers to clocked Boolean design as the fundamental referent with asynchronous design as a subordinate derivative variation. The present narrative takes the opposite view considering dependently flowing self coordinating behavior as fundamental and synchronous clocked behavior as derivative. Closure, with its counter flowing network, presents a more integrated wholeness of expression than does the notion of local handshake control.

