DISTINCTION NETWORKS

William Bricken
July 1995

Abstract. Intelligent systems can be modeled by organizationally closed networks of interacting agents. An interesting step in the evolution from agents to systems of agents is to approach logic itself as a system of autonomous elementary processes called distinctions. *Distinction networks* are directed acyclic graphs in which links represent logical implication and nodes are autonomous agents which act in response to changes in their local environment of connectivity. Asynchronous communication of local decisions produces global computational results without global coordination. *Biological/environmental programming* uses environmental semantics, spatial syntax, and boundary transformation to produce strongly parallel logical deduction.

1 Definitions

Intelligent software agents are computer programs which operate with moderate to high autonomy within software environments to perform useful, comprehendible computation. The type of computation an agent might perform is often described in terms of human needs, activities such as goal directed information retrieval, strategic and tactical planning, resource coordination, mail filtering, tool interoperability, and human friendly interface. An agent is expected to engage and help the user [16], so agent interface and modeling is often formulated in anthropomorphic terms. Agents are described, for example, as running a sense-compute-act loop, simulating the perceptual-cognitive-behavioral process of a living thing. Sensing and acting directly implicate an environment within which the agent is embedded.

Intelligent software is software which has some variant of the predicate calculus at its mathematical origins. This consideration is not stringent, but suggests both a formal basis and a pattern-matching capability underlying the agent model.

Agent systems are ensembles of software agents which are endowed with relevant network connectivity. In contrast to distributed computation and neural networks, agent systems incorporate two additional constraints: no global control of agent behavior, and semantics associated with connectivity itself. The *local decision-making constraint* places an emphasis on asynchronous multiple agent coordination. The *semantic connectivity constraint* requires that the network topology be non-trivial, that the communication channels between agents be meaningful to the interconnected agents, in effect defining a relevant environment.

Emergent properties characterize the global activity of an ensemble without being defined by any member of the ensemble. (The ensemble concept originated from thermodynamic laws which are not embodied in the behavior of any single constituent molecule.)

The *biological/environmental metaphor* [4, 1] introduces several systems-oriented modeling techniques, such as situated behavior in first-class environments, non-determinism, organizational closure, autonomous self-modification, and emergent properties [see 20]. Agents thus require a *systems-oriented programming* model rather than object-oriented programming. Agents communicate across a diversity of hardware, software and human resources, just as operating systems coordinate across various file systems, memory, processors and i/o devices, and just as animals negotiate a diverse and unpredictable environment.

1.1 Problem

Problems are often clarified when formulated in a minimal model. What then is a *simple* intelligent agent system? What toy problem captures the formal interaction between networked autonomous agents with very simple processing capabilities, a problem from which useful, comprehendible computational results emerge from local biologically modeled communication situated within a heterogeneous environment? This paper presents such a simple intelligent agent system (distinction networks) for the purpose of exemplifying and understanding the tools and issues introduced when constructing systems of agents.

A *distinction network* (dnet) is a directed acyclic graph of distinction nodes. Each *distinction node* (dnode) is organizationally identical, with a single *disposition*, or motivation: to disconnect from the dnet. Communication between nodes is restricted solely to messages announcing or requesting a change in the network connectivity, that is, a change in the environment. The advantage of agents which prefer not to compute is that computational overhead is kept to a minimum while comprehensibility is not overshadowed by complexity.

In order to associate an interesting mathematical functionality with dnet activity, the semantics of any Boolean function can be mapped onto network connectivity. Each link between nodes then represents a logical implication. Dnets evaluate their characteristic Boolean function in parallel through the local, asynchronous disconnection of dnodes. Dnets are similar to logic circuits composed of NOR gates, but do not contain coordination structures such as clocks, blocking, or request/acknowledge.

In dnets, local communication between minimal autonomous agents produces global computational results such as Boolean evaluation, program control, tautology checking, expert system optimization, logic synthesis, and other useful applications of propositional calculus. Dnets serve as a prototypical example of how to structure intelligent computation as a *strongly parallel process*, i.e. one in which local, asynchronous, autonomous, partial decisions lead to valid, useful global behavior.

1.2 Representation

Traditional mathematical and programming languages have a binary sequential syntax. A logical form such as *modus ponens*,

((if (a and (if a then b)) then b) is-valid (forall a and b))

uses parentheses to group operators with arguments and to sequence computation. The two assumptions of

1) binary operations on ordered arguments and
2) ordered operator application

are embedded in the syntax, and then revoked by rules of associativity, commutativity, and distribution.

If we were to begin with a commutative group structure, as is the case for Boolean algebra, then a representation could inherently assume associativity and commutativity, with rules revoking these assumptions for structures that are non-associative or non-commutative. If we were to express propositional logic with a single operator as its basis, then distribution rules which control the ordering of logical operations would reduce to the assumed associativity rules. These two design changes in representation remove the sequential assumptions embedded in the symbolic patterns of logic. A plausible computational model which exhibits strong logical parallelism then, might include a single-operator logical notation which treats arguments as unstructured sets or multi-sets [7, 8]. Venn diagrams are an example, but they lack dynamics. Animated graphs rendered in two or three dimensions, however, provide a representational substrate for visually parallel display of set-based parallel activity.

The sequel calls on the known map from logic to distinctions [18] to simplify and non-linearize logical notation. Then distinctions are mapped onto animated graphs which display sets of autonomous proto-logical processes as graph reduction transformations. The final configuration of the graph represents the logical conclusion.

A *distinction* is a boundary which distinguishes between two sides. Conventionally, the side that the observer (or reader) of the distinction is on is named the *outside*, or *context*, while the other side is the *inside*, or *content*. The dimensionality of representation for distinctions is a design choice. On a line, a point can distinguish context from content, so long as there is an origin which orients which side of the line is which. More familiarly, a closed curve on the plane distinguishes inside from outside. The orientation is deeply conventional: the outside is associated with the observer, since the plane of display itself (the representational space) is almost always closed within our field of view. In three dimensions, rooms, boxes and other containers serve as examples of spatial distinctions.

An important property of distinctions is that changing the observer's perspective can change the origin convention and thus change our model of content and context. This permits the distinction to be overloaded with both an objective and an environmental interpretation without loss of rigor.

Distinctions provide a rich vocabulary for modeling observers as perspectives, environments as contexts, and representations as contents. The concept was first introduced in *Laws of Form* [18]. Conclusions as to the utility of this book are varied [9, 14, 15, 21]. Those who see it as "...simply another axiomatization of Boolean algebra" [10] may have fundamentally misunderstood the mathematics itself, in essence confusing the void with a representation of the void (the classical General Semantics error). Those who understand it as "... not an arbitrary new calculus, but that particular calculus which can let us see deeper into the nature of mathematics" [22] are uniformly convinced of its potential utility.

1.3 Void

Distinctions are constructed in an empty context, *a representational space*. Drawing a closed loop on the page, for example, indicates a distinction, apparently cleaving the page into two parts, an inside and an outside. The page itself, however, is not torn asunder, it remains whole. Distinctions do not interact with their substrate. The representative space has no metric, it is devoid of characteristics and thus transparent to operations on distinctions. The representational space *pervades* all distinctions it contains; it is both the outside and the inside. Pervasiveness means that a distinction does not create a Cartesian duality; context and content are not separated by EXCLUSIVE-OR, they are associated by INCLUSIVE-OR.

The representational space provides a unique tool which is not present in traditional notations: it can be *void* (unmarked or empty). The void cannot be accessed directly, since it is not perceptible. Distinctions provide an indirect access: an *empty distinction* indicates a void content. Since the void can be indicated, it can be used semantically, to (non)represent concepts in the modeled domain.

Boundary mathematics is based on representations of distinctions as containers. Its origin concept is the void. The introduction of a useful void is analogous to the introduction of a useful zero in number systems. Just as zero overcomes a weakness in Roman numerals by permitting efficient algorithms for multiplication, the void overcomes a weakness in logical notation by permitting efficient algorithms for deduction.

1.4 Boundary Logic

This section describes the mapping between distinctions and propositional logic. The objective is to construct a network-based parallel logic which provides an elementary example of agent systems.

Parens notation represents configurations of distinctions as nested (( )) and concatenated ( ) ( ) parentheses. It provides two elementary syntactic structures which can be used to represent logical operators: distinction ( ) and void .

One possible minimal basis for propositional logic is the set {IF, FALSE}: all other logical operators can be constructed from this set. Table 1 maps the logical operator IF onto the distinction, and the logical constant FALSE onto the void. IF A THEN B is represented as a boundary between the antecedent and the consequent. FALSE has no representation, since the void is not (and cannot be) represented. The basis for expressing propositional logic as boundaries is thus a singleton set { ( ) }.

TRUE	( )
FALSE	
IF *a* THEN *b*	(*a*) *b*
NOT *a*	(*a*)
a OR *b*	*a* *b*
a AND *b*	((*a*) (*b*))

Table 1. The Map from Propositional Logic to Parens Boundary Logic

Table 1 presents several other logical operators in parens form. The map from propositional logic to boundary logic is many-to-one; boundary logic is formally morphic to Boolean logic, but it is *not isomorphic*. Since classes of Boolean logic expressions map onto individual elements in boundary logic, both representation and computation are simpler using distinctions. Conversely, boundary configurations can be interpreted for logic in many ways. An empty parens, for example, can be read declaratively as the constant TRUE, or as the unary operation of negation NOT FALSE, or as the binary operation of implication, IF FALSE THEN FALSE. Alternatively, the distinction can be seen as a generalized form of NOR. With no arguments, it is NOR FALSE = TRUE. With one argument, it is negation, NOR A = NOT A; with two or more arguments it is n-ary NOR. Forms sharing a representational space are not grouped or ordered, since space supports no structure. From a boundary perspective, sharing space is a set operator. The functional interpretation of space for logic is generalized n-ary OR. That is, the void is overloaded with both the OR operator and the constant FALSE.

Boundary logic is an axiomatic system based on distinctions interpreted for logic. Table 2 presents a three axiom basis for propositional boundary logic. The system emphasizes formal transformation by void-substitution, in effect erasing irrelevant symbols. Thus any boundary structure with the pattern (A ( )) is assured by DOMINION to be irrelevant to the outcome of the computation in that pattern's context, or environment. The DOMINION rule gives permission to disregard all such patterns by permitting void-substitution for those patterns. INVOLUTION disregards a particular double boundary structure, while PERVASION disregards duplicate reference to subexpressions. These axioms are independent, complete and consistent. They also provide a particularly simple computational model, since the right-hand-side of each rule incorporates a void-substitution.

Dominion	(a ( )) =	
To ABSORB: Erase distinctions that contain an empty distinction, and their contents.		
Involution	((a)) = a	
To CLARIFY: Erase distinctions with no intermediate structures.		
Pervasion	(a (a b)) = (a (b))	
To EXTRACT: Erase duplicates of the context from the content.		

Table 2. The Axioms of Boundary Logic in Parens Notation

A sample proof of the idempotency of OR follows:

a	a = a	IDEMPOTENCY
a	a	given
((a))	a	involution
(( ))	a	pervasion
	a	involution

This proof incorporates several characteristics of the boundary mathematics formalism: an algebraic transformation strategy, extraction of redundant forms from deeper nested spaces, and transformation by void-substitution rather than by rearrangement.

2 Distinction Networks

The structure, or connectivity, of a distinction network embodies the logical semantics of its Boolean function. Each dnode represents a distinction which acts only on its local context as determined by its upper and lower connectivity. When dnodes are implemented as agents with the disposition of the generalized NOR operator, they act autonomously and in parallel to minimize their local connectivity. The activity of the ensemble generates logical deduction.

Figure 1 illustrates the spatial translation process from logic to parens to distinction networks. The linear parens representation is first spread out over a plane, associating a spatial extent with the depth of parens nesting. Each parens pair defines the boundary of a distinction node; nesting of parens defines directed links between nodes. Multiple references to variables are combined so that each variable node is unique.

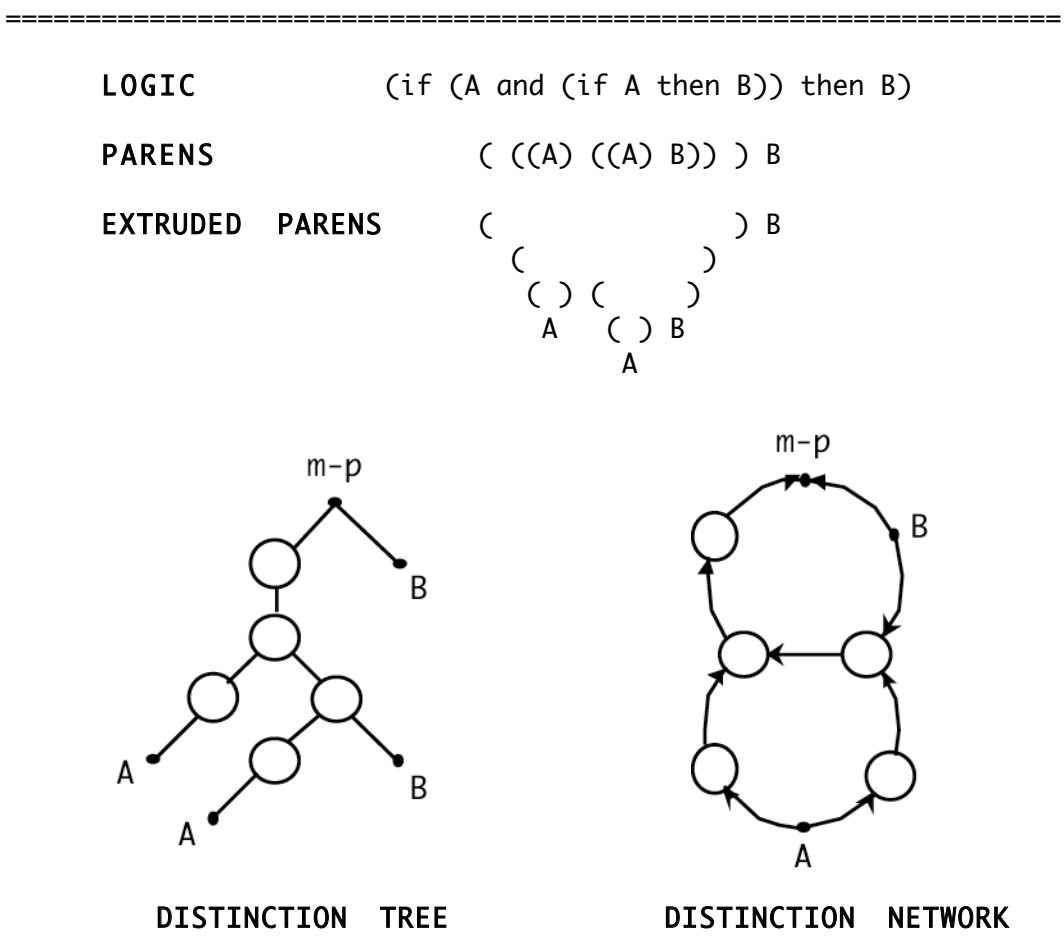

Figure 1. Translating *Modus Ponens* from Logical Syntax to Dnets

Figure 2 illustrates the three reduction rules of boundary logic in distinction network form. Capital letters stand for arbitrary subnetworks, including none or several. The middle forms in each reduction rule include a *disconnect notation* which illustrates how dnets are transformed during a reduction. The axiomatic transformations constitute a *graph rewrite system*.

The next section illustrates logical evaluation using a single reduction rule expressed in an agent-oriented model and a pseudo-LISP syntax. Algebraic minimization follows, incorporating the other two axioms into the agent model.

2.1 Structure and Organization of Dnets

Dnets provide two perspectives on computation. From the standard outside perspective, the organization of the network is that of communicating distinctions. The structure of the network is its particular connection topology, which represents its Boolean functionality. The orientation of the network is from ground input to global output, with the upper bound representing ensemble functionality and the lower bounds representing function parameters and variables.

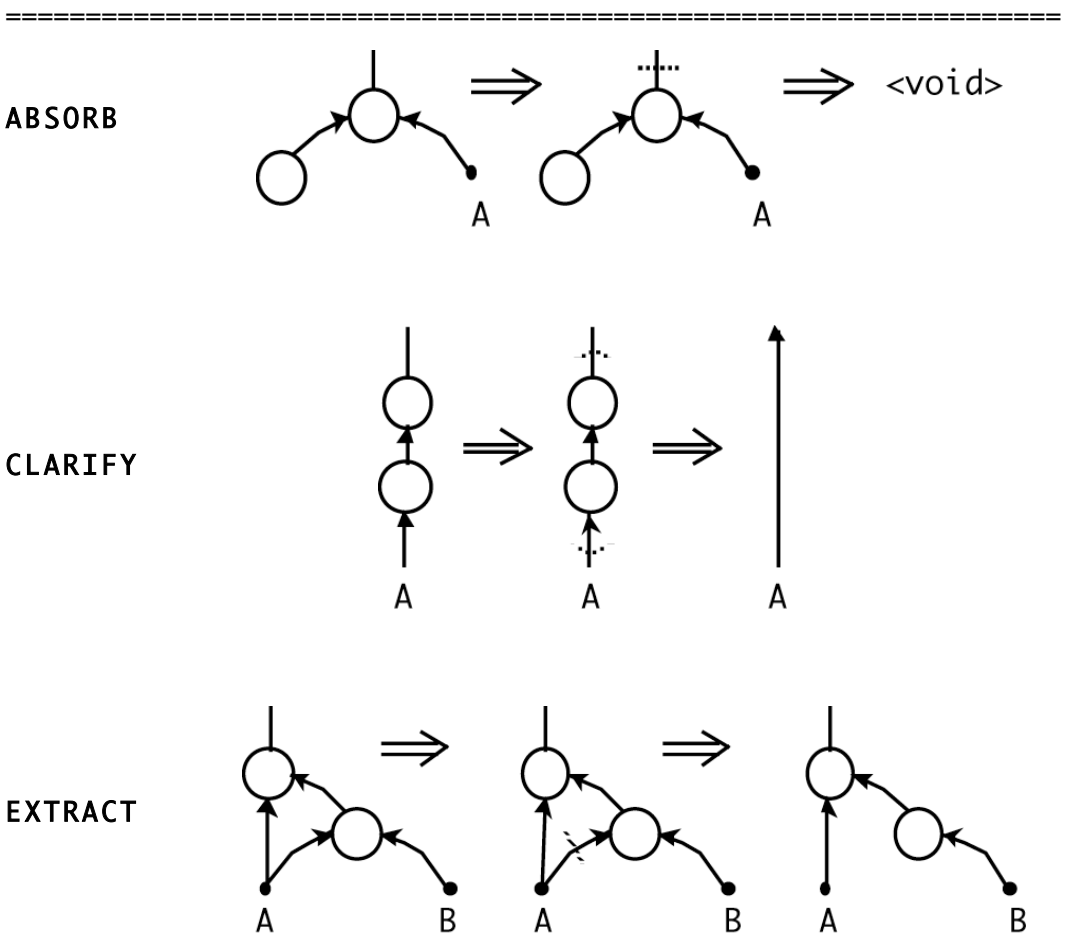

Figure 2. Boundary Logic Reduction Rules in Dnet Notation

The non-standard perspective defines organization and structure from within each individual dnode while it senses and reacts to its local environment. The agent-oriented model of a dnode with the disposition to evaluate logic is presented in Table 3.

The internal organization of each dnode consists of a local memory which represents the sensed environment (UPPER, LOWERS), two reactive message handlers (ABSORB!, REMOVE!) which result in immediate actions, and one persistent internal process (ABSORB?) which generates an action only in the particular environmental circumstance of no lowers. This regime implements the boundary logic reduction rule ABSORB. ABSORB eliminates all occurrences of the empty parens. It is a graph generalization of partial function evaluation in the Boolean domain.

The internal structure of different dnodes differs only in the contents of their local memory. Local memory stores a dnode's only perceptual involvement with its environment as the names and orientations of its direct communication partners. Messages with a "?" suffix indicate a choice local to a single dnode based on its memory of its current environment. Messages with a "!" suffix are reactive, requiring no contextual or internal evaluation (in different metaphors: no awareness of situation, no memory-based computation,

no contemplation).

To ABSORB: (A ( )) ==>
Local memory (knowledge of environment):
UPPER <the upper node linked to a dnode> LOWERS <lower nodes linked to a dnode>
Disposition and initialization:
ABSORB? (if (no lowers) then (send upper ABSORB!))
Response to messages:
ABSORB! (send upper REMOVE! self) REMOVE! <link> (forget link)

Table 3. Dnode Organization for ABSORB

Many implementation details have been omitted from this organizational description. For instance, no attempt has been made to clean up dnodes that are left stranded; it is sufficient to note that dropped messages and stranded nodes will not effect the validity of the global outcome. Also, no provision has been made for the behavior of variable nodes which may be bound to a value or left unevaluated.

When a node disconnects from (forgets) its upper, it effectively removes itself and its subnetwork from further interaction with the active ensemble. Lower-bound exceptions (for example, unbound variables have no lowers yet do not ABSORB) and upper-bound exceptions (do not CLARIFY the observer node) define the edges of a dnet for i/o purposes. Labeled upper and lower bounds are the observable edges of the ensemble, and begin half disconnected.

Consider the dnet in Figure 3, which represents the functionality of a one-bit equality tester (the IFF function). When an input variable is bound to TRUE, it is replaced by a dnode. When a variable is bound to FALSE, it is erased (equated to the void). In the example, when A is TRUE and B is FALSE, the value of the IFF function is FALSE. Variable binding provides the first reduction, as A and B substitute their current values. Two ABSORBs from A=( ) are the second step. The three graph reduction steps illustrate one possible evaluation path, which is summarized in parens form below.

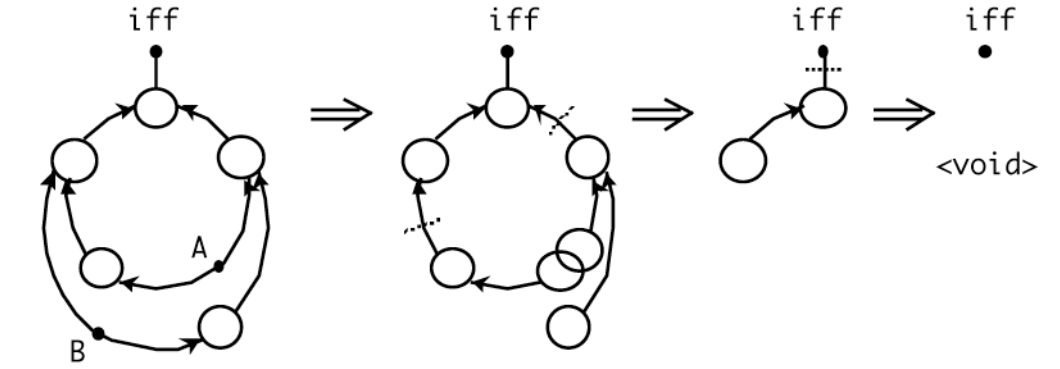

Figure 3. Dnet Evaluation of the IFF Function.

```
( (( a ) b) ( a (b)) )          iff function
( ((( ))  ) (( )( )) )          a=( ), b=<void>
( (       )        )            absorb (()), (()())
       <void>                   absorb (())
```

Differing arrival times of messages at different dnodes lead to different, but equifinal, evaluation paths. Two other possible paths are presented below in parens notation:

```
( ((( ))  ) (( )( )) )       bind variables
( (       )        )         absorb (()),(()())
       <void>                 absorb (())

( ((( ))  ) (( )( )) )       bind variables
( (       ) (( )( )) )       absorb (())
       <void>                 absorb (()(()()))
```

2.2 Algebraic Reduction

Agent-oriented pseudo-code for the ABSORB transformation is presented in Table 3. The code for the two other reduction rules, CLARIFY and EXTRACT, is presented in Tables 4 and 5. The only autonomous aspect of a dnode's disposition during Boolean evaluation is the self-query ABSORB?. Otherwise the network behavior is purely reactive, or data-driven. A more complex form of coordinated agent behavior occurs during algebraic reduction of dnets, when variables are not bound. In algebraic reduction, dnodes cooperate to enact all three reduction axioms of boundary logic, generating partial function evaluation, tautology identification, and Boolean minimization.

The CLARIFY reduction rule removes two dnodes which have no intermediate network structure. It is a graph generalization of functional inversion. The CLARIFY? message coordinates with the lower (deeper) node of the two. The CLARIFY! response assures the upper node that there is no intermediate network structure between the pair, and that the lower node has not already entered into another CLARIFY relationship as an upper, as occurs in the structure

```
( ( (A) ) ) = (A)
```

Rather than dynamically rearrange links, the two dnodes in this implementation turn into communication channels, propagating information from the content of the lower node to the context of the upper node, while rendering their own disposition inert.

To CLARIFY: ((A)) ==> A
Disposition and initialization: CLARIFY?? (if (one lower) then (send lower CLARIFY?))
Response to messages: CLARIFY? (if (>1 lowers)) then ((send upper CLARIFY! lowers) and (become-a-link))) CLARIFY! <links> ((send upper JOIN! links) and (become-a-link)) JOIN! <links> (perceive-as-lower links)

Table 4. Dnode Organization for CLARIFY

In the EXTRACT reduction rule, each dnode with lowers propagates an instruction downward to all nodes in its subnet to disconnect from those lowers. This action is a graph generalization of deduction from the excluded middle in logic, optimization of scoping in programming, and proper subset in mathematics. EXTRACT is a deep rule, applying regardless of the depth of nesting of a target variable. Thus,

```
          a (b (c (a d) ) )
==>       a (b (c (  d) ) )
```

In the implementation in Table 5, all dnodes with lowers independently issue EXTRACT!, flooding the network with downward propagating messages. Alternatively an implementation could trade space for time, and propagate only one EXTRACT! message from the upper bound of the network while dynamically accumulating potentially redundant lowers in each subnetwork.

To EXTRACT: (A (A B)) ==> (A (B))
Disposition and initialization:
EXTRACT? (if (>1 lowers) then (send lowers EXTRACT! lowers))
Response to messages:
EXTRACT! <links> ((if (link = lower) then (forget lower)) and (send lowers EXTRACT! links+lowers))

Table 5. Dnode Organization for EXTRACT

Figure 4 demonstrates an algebraic proof of *modus ponens* (m-p). The translation of m-p to parens and to dnets is in Figure 1. In Figure 4, the dnode-agents communicate while making local decisions. Two nodes CLARIFY while one link is EXTRACTed from B. Three sequential reductions then terminate the process with m-p = ( ). The parens proof, which ends at identity, follows:

```
(((a) ((a) b))) b = ( )          modus-ponens
  (a) ((a)    ) b = ( )          clarify, extract b
  (a)   a       b = ( )          clarify
  ( )   a       b = ( )          extract a
  ( )              = ( )          absorb a b. identity
```

2.3 Dnet Dynamics

Dnodes without lowers propagate ABSORB! upwards while dnodes with one lower request CLARIFY? downwards and dnodes with more than one lower propagate EXTRACT! downwards. This models, respectively, concurrent data-driven, structure-driven, and goal-driven graph reduction strategies. The three axioms can be implemented to integrate seamlessly into an ensemble of parallel activity. From the perspective of any single dnode, it is acting out its own disposition within a changing environment. From the perspective of the entire dnet, a swarm of local activity transforms the net into a minimal graph representation of its Boolean function. This local activity does exhibit global patterns, beginning with all ABSORB and CLARIFY transformations, then alternating between waves of EXTRACT and local ABSORB/CLARIFY in response. The process terminates when all EXTRACTs fail.

Technically, more mechanism is needed to achieve complete Boolean minimization, since dnets can settle into local, non-global minima. As well, the concept of minimization itself is goal dependent, requiring different mechanism for different objectives. For example, in logic synthesis for circuitry, depth of nesting models the time delay of propagation of a signal,

the number of links in a network models wiring costs, and the number of dnodes models layout area. Different silicon technologies emphasize different concepts of optimality. Mathematically, the goal is to maintain functional invariance across network transformations while meeting design constraints.

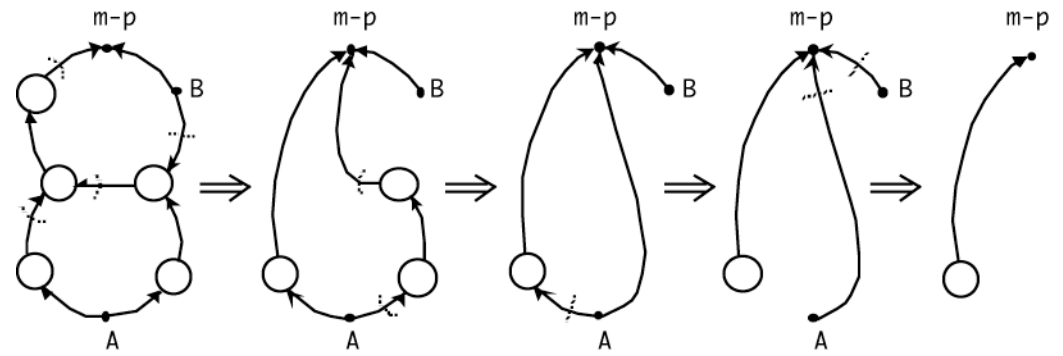

Figure 4. Dnet Graph Reduction of the *Modus Ponens* Tautology

For simplicity, the above implementation assumes one upper connection for each dnode, with the exception that ground variables have an arbitrary number of uppers. An additional structure-driven graph transformation (COALESCE: A A ==> A) condenses the number of nodes in a dnet by converting multiple representation of entire subnets into multiple reference to a single subnet. This generalizes the structure of dnets to include dnodes with multiple uppers.

3 Applications

With implementation detail and with specific performance extensions, boundary mathematics has been tested extensively for modeling programming languages [6], for visual user interface [3], for logical deduction in expert systems [5], and for circuit synthesis. Systems of distinctions have been used not only for modeling propositional logic, but also for modeling predicate calculus and the mathematical domains of integer and transcendental arithmetic [2, 11], knot theory [13], physical theory [17], and self-reference and complex Boolean values [12, 19]. Naturally the primary difficulty with working with boundary mathematics is its novelty.

4 Discussion and Summary

Distinction networks are a simple example of the evolution of software models from intelligent agents to intelligent systems. The above description of dnets is intended as a brief sketch of systems of agents which, in ensemble, perform mathematical computation, and thus might be considered to act intelligently. The important aspect of the example is that the intelligence is truly distributed across the network, no agent is performing a traditional mathematical operation, and no agent has a model of the objectives of the

mathematical process. This permits each agent to act independently and without synchronization.

In order to map mathematical semantics onto networks, the existence of connectivity between nodes must be principled. In boundary logic, connectivity is defined by containment relations, and containment is interpreted as implication. The fundamental innovation in boundary mathematics is the use of higher dimensional representations such as containers which permit a semantic interpretation of the absence of representation (the void). Like electronic circuits, a distinction network can define a particular logical function. Unlike circuits, the signal traveling along enabled communication channels represents a local exchange about the relative context of each node rather than a logical value. Signals are thus decoupled from logical semantics and serve purely to communicate the actions of nodes to their neighbors. Nodes can then be viewed as enacting their dispositions autonomously in response to changing environmental circumstances.

The biological/environmental model for programming agents is enabled by the techniques of:

- *Environmental semantics:* distributing a problem over an ensemble of agents by mapping its structure onto local connectivity between agents,

- *Spatial syntax:* representing a problem in a notation which supports principled topological transformations, spatial display and direct interaction, and

- *Boundary transformation:* utilizing void-substitution to remove irrelevant parts of a problem.

Boundary mathematics provides new conceptual tools, such as <void>, representational space, distinction, observer perspective, pervasion, ensemble, and environment, which permit modeling strongly parallel problem solving with locally coordinated systems of simple agents.

References

[1] R. Beer, H. Chiel and L. Sterling (1990) A biological perspective on autonomous agent design. In P. Maes (ed.): *Designing Autonomous Agents*, MIT Press, 169-186.

[2] W. Bricken (1992) Spatial representation of elementary algebra. *1992 Visual Languages*, IEEE Press, 56-62.

[3] W. Bricken (1988) Computational drawings. *Autodesk Research Lab Presentation*.

[4] W. Bricken and G. Coco (1995) VEOS: the virtual environment operating shell. In W. Barfield and T. Furness (eds.): *Virtual Environments and Advanced Interface Design*, Oxford Univ. Press, 102-142.

[5] W. Bricken and E. Gullichsen (1989) An introduction to boundary logic with the Losp deductive engine. *Future Computing Systems*, **2**(4), 1-77.

[6] W. Bricken and P. Nelson (1986) Pure LISP as a network of systems. *Proceedings of the Second Kansas Conference: Knowledge-Based Software Development*, Kansas State University.

[7] A. Bundy (1983) *The Computer Modeling of Mathematical Reasoning*, Academic Press.

[8] W.F. Clocksin and C.S. Mellish (1981) *Programming in Prolog*, Springer-Verlag.

[9] P. Cull and W. Frank (1979) Flaws of form. *Int. J. General Systems*, **5**, 201-211.

[10] W.E. Gould (1977) Review of laws of form. *J. Symbolic Logic*, **42**, 317-318.

[11] J. James (1993) A calculus of number based on spatial forms. M.S.E. Thesis, University of Washington.

[12] L.H. Kauffman (1987) Self-reference and recursive forms. *J. Social and Biological Structures,* **10**, 53-72.

[13] L.H. Kauffman (1994) Knot automata. *24th Int. Symposium on Multiple-valued Logic*, IEEE Press, 328-333.

[14] L.H. Kauffman and F.J. Varela (1980) Form dynamics. *J. Social and Biological Sciences*, **3**, 171-206.

[15] L.J. Kohout and V. Pinkava (1980) The algebraic structure of the Spencer-Brown and Varela calculi. *Int. J. General Systems*, **6**, 155-171.

[16] D. Riecken (1994) Intelligent agents. *CACM*, **37**(7), 18-21.

[17] R.G. Shoup (1992) A complex logic for computation with simple interpretations for physics. *PhysComp'92: Workshop on Physics and Computation*, IEEE Press.

[18] G. Spencer Brown (1969) *Laws of Form*, Bantum.

[19] F.J. Varela (1975) A calculus for self-reference. *Int. J. General Systems*, **2**, 5-24.

[20] F. Varela and P. Bourgine (eds.) (1992) *Toward a Practice of Autonomous Systems:* MIT Press.

[21] F.J. Varela and J.A. Goguen (1978) The arithmetic of closure. *J. Cybernetics*, **8**, 291-324.

[22] L.L. Whyte (1972) Review of laws of form. *British J. Philosophy of Science*, **23**, 291-292.