Report Number: WUCS-86-05

1986-03-01

Determinacy of Hierarchical Dataflow Model

Takayuki Dan Kimura

A parallel computation model suitable for icon based visual programming languages is proposed. The model is uses to design a functional programming language for school children. A computation is specified by boxes and arrows forming a partially ordered set of nested boxes. Loops and Boolean data tokens are eliminated from the traditional dataflow model. Block structures are logical consistency (exception) are added. A declarative semantics of the model is defined formally. Using the formalism it is proved that the model is determinate.

DETERMINANCY OF HIERARCHICAL DATAFLOW MODEL

Takayuki Dan Kimura

WUCS-86-5

March 1986

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

This research was funded by Computer Services Corporation (CSK).

Abstract

A parallel computation model suitable for icon based visual programming languages is proposed. The model is used to design a functional programming language for school children. A computation is specified by boxes and arrows forming a partially ordered set of nested boxes. Loops and Boolean data tokens are eliminated from the traditional dataflow model. Block structures and logical consistency (exception) are added. A declarative semantics of the model is defined formally. Using the formalism it is proved that the model is determinate.

1. Introduction

This report introduces a computation model based on the concepts of dataflow and completion. The primary purpose of the model is to provide a semantic foundation for a new programming language called Show & Tell™ Language (STL) which was designed for novice computer users such as school children. STL is an icon driven programming language[2] that integrates the computer capabilities for managing computation, communication, and database in home and classroom environments. The design philosophy of the language and its implementation on Apple® Macintosh™, Macintosh Show & Tell language (MSTL), will be described in separate reports[3] [4]. This report is self-contained and requires no knowledge of STL.

Dataflow[5] was chosen for its understandability. The concurrency aspect of dataflow was secondary. There are two factors that make it easy to understand a dataflow language; two dimensional representation and value oriented computation. Due to the principle of direct object manipulation[6], a graphical object is more acceptable to school children than a textual one. The notion of variable, or equivalently the notion of state, is one of the most difficult concepts that school children have to learn for conventional Von Neumann type programming. Absence of variables in a dataflow language should make learning easier.

Dataflow also provides a good paradigm for the message based communication facilities common in most home and classroom environments. Data-driven asynchronous execution of an operation is a typical mode of interaction among school children through communication media such as

paper notes, telephone, and video. Dataflow is a natural integration mechanism for computation and communication.

A completion problem[7] is to fill in the missing portions of a partially hidden pattern in such a way that the completed pattern satisfies a set of consistency rules. Some examples of general completion problems are given in Figure 1.

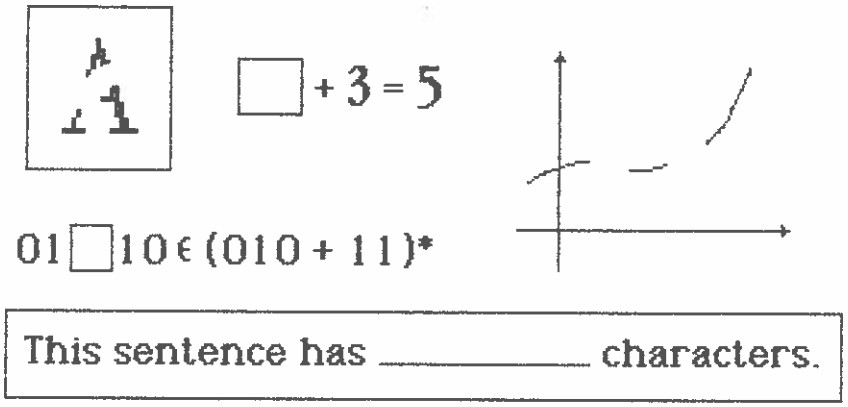


Figure 1 : General Completion Problem

Completion was chosen for its capability of integrating the notions of computation, communication and data query. An arithmetic computation is the process of solving a completion problem whose consistency rules are arithmetic (Figure 2(a)). A communication is the process of completing the missing information the receivers have with exactly the same information as the sender has. The consistency rules for such a completion are defined by whose information is to be shared with whom (Figure 2(b)). A data query is a completion problem where consistency rules are defined by membership in a set of records in a file (Figure 2(c)).

The computational completion problem can be solved by execution of arithmetic operations, the communication problem by transmission of messages, and the query completion problem by execution of pattern matching operations. Thus, completion is a unifying concept for computation, communication and data query. The Show & Tell language is designed as a specification language for such completion problems. However, in this report we will concentrate on computational aspect of the completion problem.

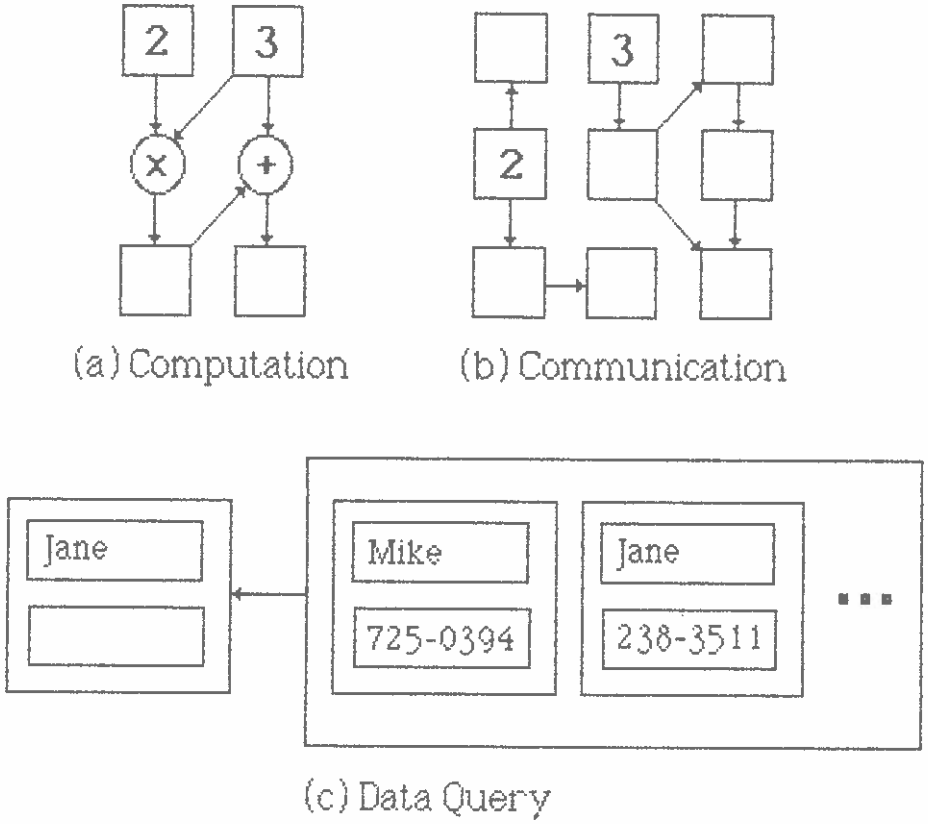


Figure 2 : Completion as Integration Mechanism

The dataflow model by itself is not an appropriate computation model for school children. First of all, a cycle (loop) in a dataflow graph requires an

introduction of the notion of state, which makes learning the semantics more difficult for children. Secondly, there is no abstraction mechanism inherent in the model by which complex problems can be decomposed into simpler ones. By the same token, there is no construct similar to the begin-end block of ALGOL, well-known for its effectiveness in program structuring. Thirdly, the dataflow switching operations, such as distributors and selectors, are not high level enough for children who are not familiar with hardware related concepts. Representing the result of decision making by a data token input to a switching operation is counter-intuitive, and the integration of control flow with data flow is not appropriate for novice programmers. Finally, there is no encapsulation mechanism by which error propagation can be controlled and modular programming can be exercised.

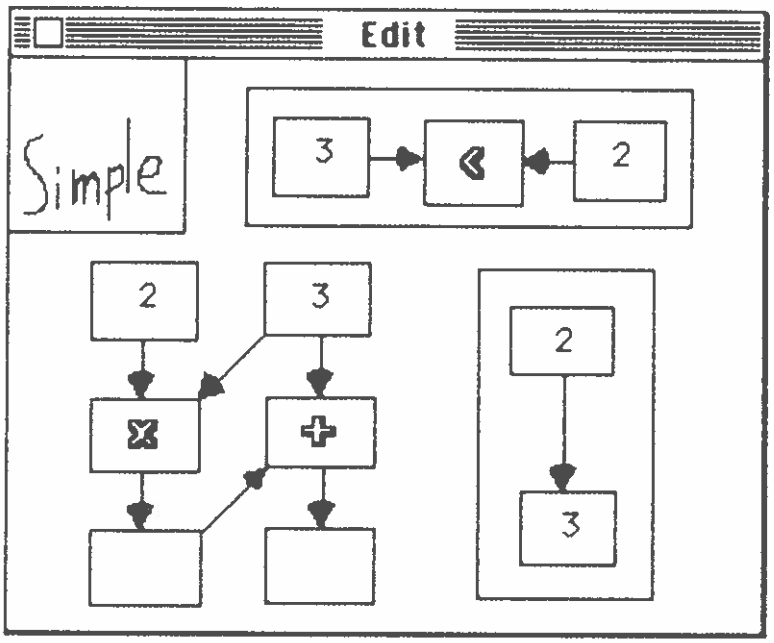
In order to resolve the above difficulties, a new computation model was developed by introducing the concepts of block structure and logical consistency into the traditional dataflow model. We call the model a hierarchical dataflow model (HDM). In the next section we will describe the model informally with examples using the STL syntax. Then, the formal definition will be given. Finally, the determinancy of the new model will be shown.

2. Boxes and Arrows

In the hierarchical dataflow model (HDM) a computation is specified by a box-graph, a set of boxes connected by a set of arrows. No cycle or loop is allowed in a box-graph. Each box may be empty or may contain a data element, an operation, a predicate, or another box-graph. Boxes can be nested. An arrow directs the flow of data from one box to another, and defines the consistency relationship between the boxes. Each arrow has exactly one starting box and one destination box. Arrows do not branch out.

A box-graph is consistent if there is no conflict, directly or indirectly, among the contents of the boxes; otherwise it is inconsistent. For example, a box-graph is inconsistent if there are two boxes containing different data values and the boxes are directly connected by an arrow. A copy of data value in the starting box of an arrow moves to the destination box, unless the destination box already contains a different data value. If it does, then the entire box-graph containing these boxes is inconsistent.

The set of empty boxes in a box-graph is called the base of the graph. A computation in HDM is to fill the base of a box-graph with a set of data in such a way that the completed graph is consistent. It involves both finding data to fill the base with and testing the consistency of the box-graph. Once a base box is filled, the content of the box never changes. Note that higher level objects such as operations and predicates cannot be used to fill empty boxes in HDM. Figure 3 (a) gives a simple computation problem consisting of three independent components; one to fill in empty boxes and two for testing inconsistencies. Figure 3 (b) displays a solution obtained by the MSTL system.



(a) Simple Box-Graph before Execution

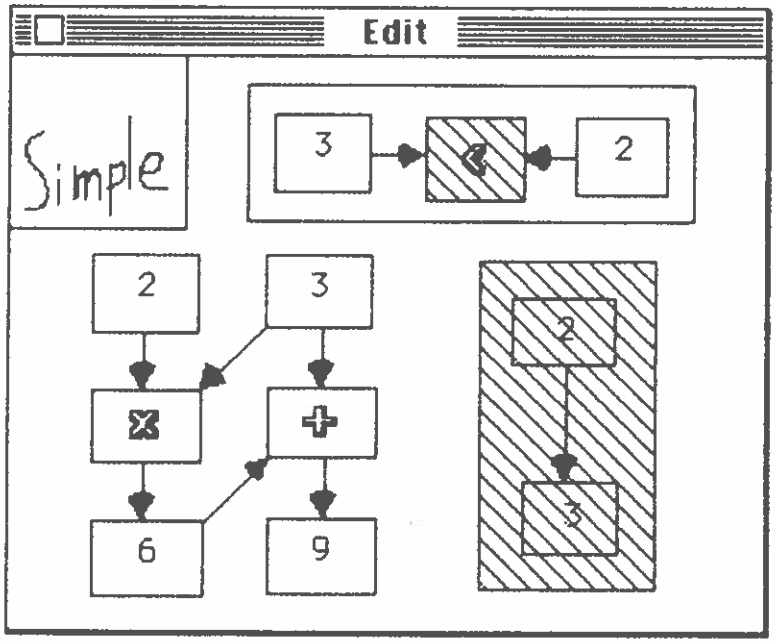
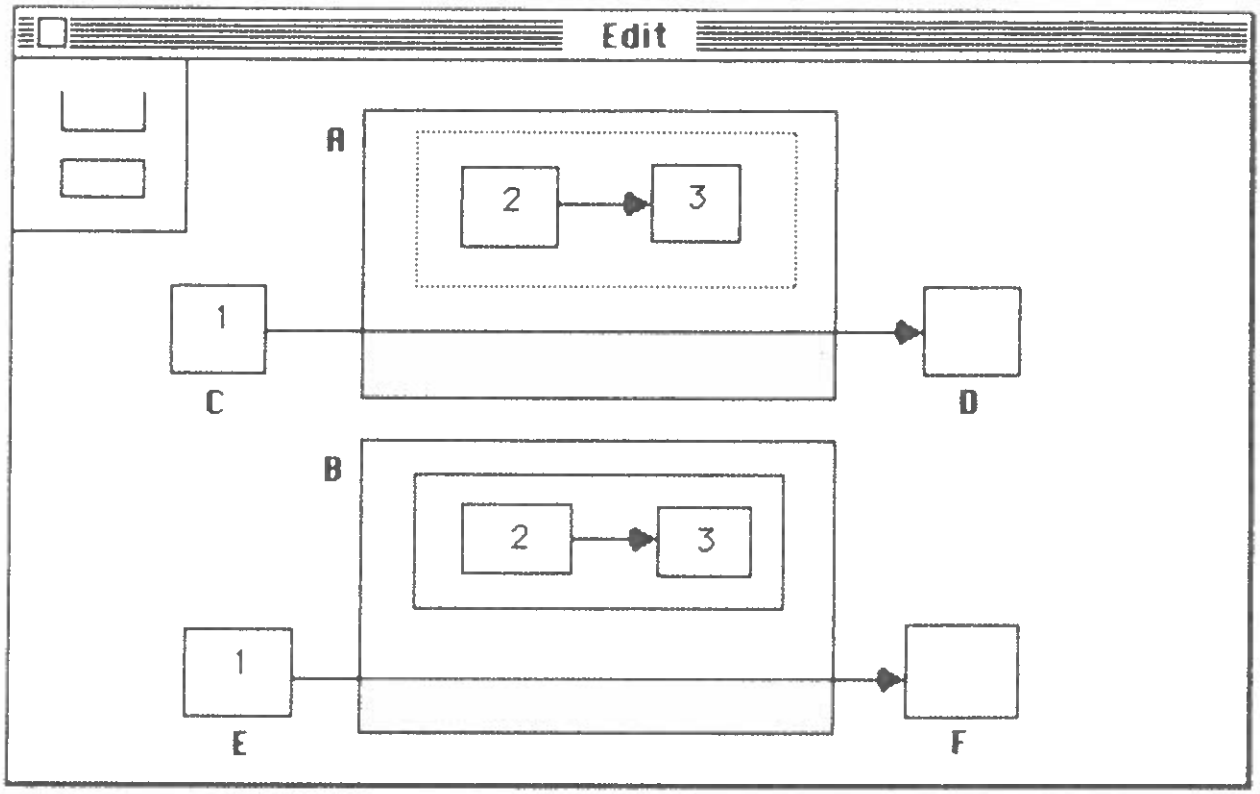


Figure3: (b) Simple Box-Graph after Execution

All inconsistent box-graphs are shaded by the system when solutions are derived. Note that the predicate '<' in MSTL is a system defined box-graph, therefore they can be either consistent or inconsistent. The square on the left-upper corner contains a user defined name (icon) of the box-graph drawn inside the window. The icon name stands for the box-graph when it is used in another box-graph. Such an icon name corresponds to a subroutine name in traditional programming.

A box containing a box-graph is called a complex box. A complex box can be open, represented in MSTL by a dash-line rectangle, or closed, represented by a solid-line rectangle as illustrated in Figure 4. If an open box contains a box-graph which is inconsistent, then the box-graph containing the open box is also inconsistent by definition, i.e., inconsistency propagates through the boundary of an open box to the larger context. If a closed box contains an inconsistent box-graph, then the closed box becomes non-existent, i.e., the box with its contents and all arrows incident with the box can be deleted from the box-graph without changing the consistency property of the graph containing the closed box. In a sense, an open box broadcasts the exceptional condition (i.e., inconsistency) of its components towards the members of its community, and a closed box delimits the boundary of such broadcasting. This mode of communication, or propagation of information, is in contrast with that of point-to-point communication represented by an arrow.

Figure 4 illustrates the differences between an open box and a closed box. The complex box A is inconsistent, because there is a conflict in "2 flowing into 3", and the conflicting part is enclosed in an open box.



(a) Open and Close Boxes before Execution

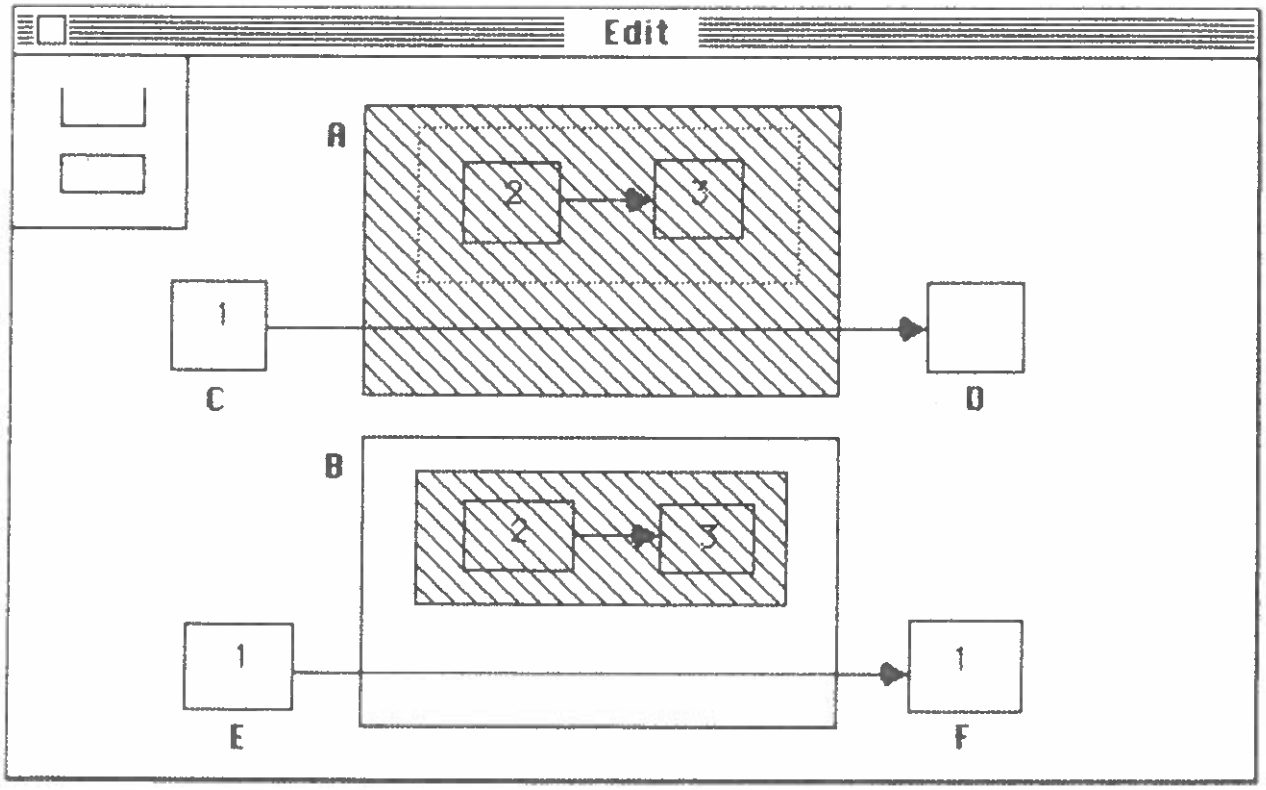
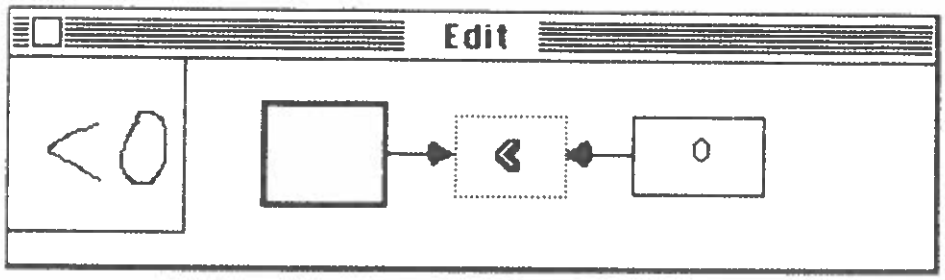


Figure 4: (b) Open and Close Boxes after Execution

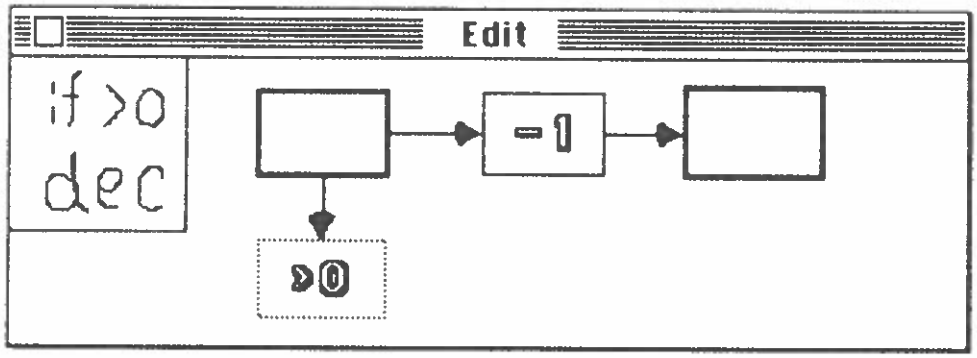
Since the inconsistent box becomes non-existing, the connection between the two boxes C and D becomes broken, and the data in C does not flow into D. On the other hand, the complex box B is consistent, because the conflicting part is enclosed in an closed box. Thus, the data in E can flow into F.

A box-graph can be used to represent a function or a predicate of a traditional programming language in the following way: Since no cycle is allowed in a box-graph, every box-graph must contain at least one component box to which no arrow points and at least one box from which no arrow starts. These boxes are called the minimums and the maximums of the graph. The empty minimums and maximums are called the in-boxes and the out-boxes of the box-graph, respectively. They represent the input and the output formal parameters of the function. When all in-boxes are filled by the argument data and the box-graph is solved, the out-boxes produce the value of the function. If there is no solution to complete the graph into a consistent one, the box-graph will be returned as inconsistent. In other words, a box-graph, representing a function, can return an exception condition as well as its value. If the box-graph does not have any out-box it defines a predicate. When and only when the predicate is satisfied by the input arguments, the graph will be returned as a consistent graph.

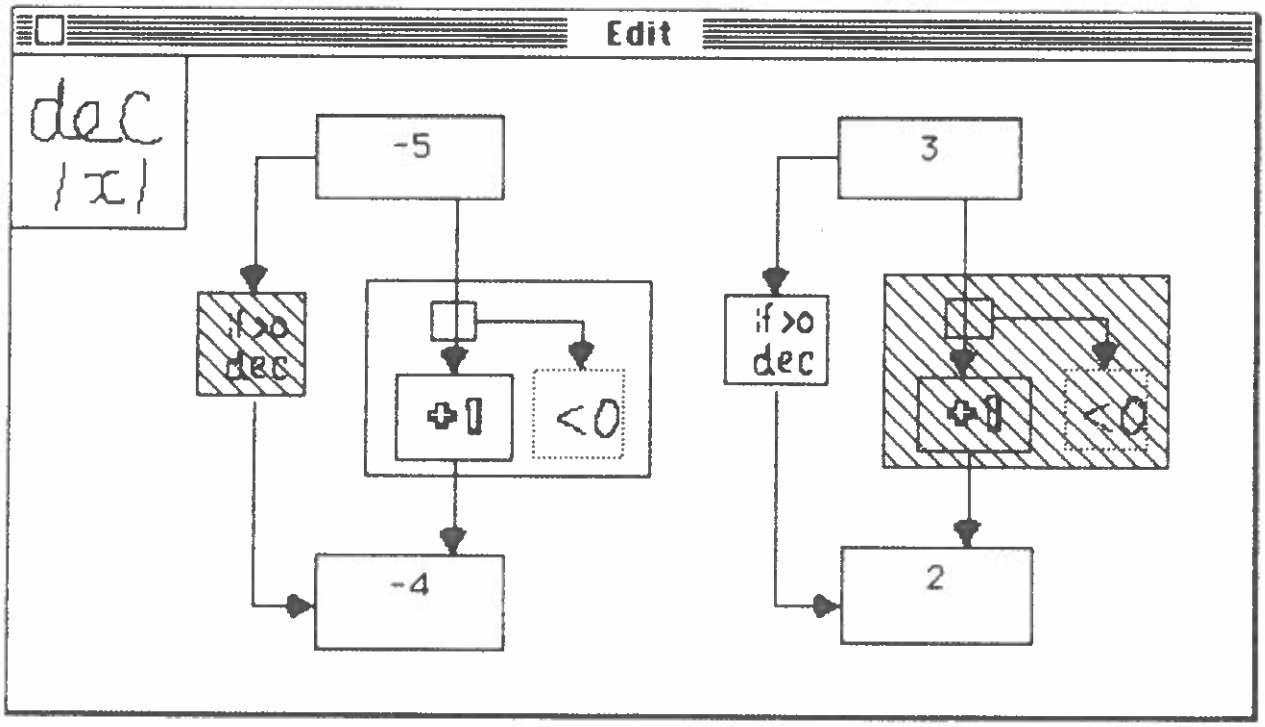
For example, Figure 5 (a) defines a predicate for testing whether a number is negative or not. Figure 5 (b) defines a function that returns the input value minus one if the input is positive, otherwise it returns the exception (inconsistency). Figure 5 (c) demonstrates how these functions can be incorporated in another box-graph. Note that in MSTL an icon name can be used to construct a complex box instead of a box-graph as its content.



(a)



(b)



(c)

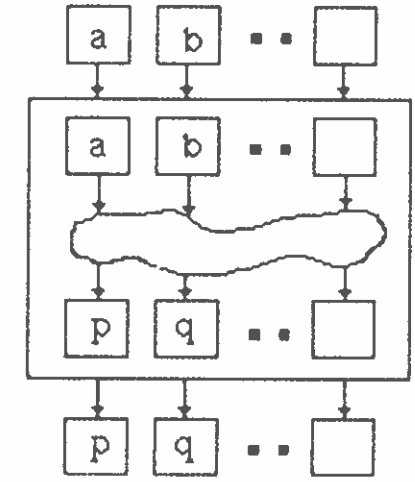
Figure 5: Function with Exception

Figure 5 (c) is the result of executing a box-graph in which the same computation is duplicated with two different input values, one positive and the other negative.

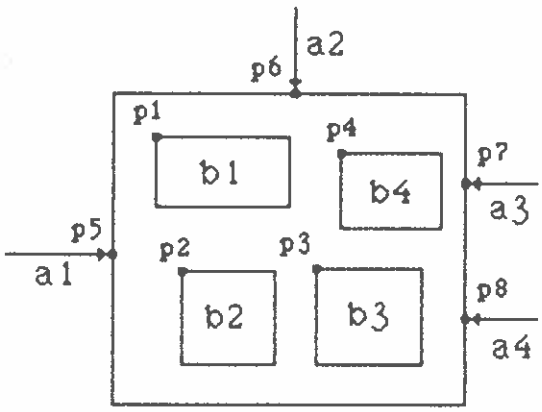
When a box-graph is contained in a complex box used as a subroutine, a positional binding rule is used to associate the arrows coming into the complex box with the in-boxes of the content box-graph. Similarly outgoing arrows are associated with the out-boxes. The binding rule is language dependent. We assume in this report that all arrows incident with the complex box can be ordered by some language specific rule. Likewise, the in-boxes and out-boxes of the content box-graph can be ordered by some rule.

Based on these orderings, the first incoming arrow will be associated with the first in-box, the second incoming arrow with the second in-box, and so on. The same holds for binding the outgoing arrows with the out-boxes. If the number of incoming arrows is different from the number of in-boxes, then the box-graph will be evaluated as inconsistent. The same is true for the outgoing arrows.

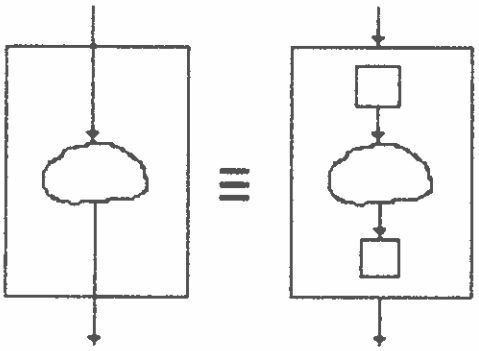
In the MSTL system, the lexicographical ordering of the (x,y) coordinate on the Macintosh screen is used for ordering boxes and arrows. The coordinate of a box is defined as that of the left-upper corner of the box, and the coordinate of an arrow incident with a complex box is defined as the intersection point with the box. The binding rule of MSTL is illustrated by Figure 6 (a) and (b). In (a), the incoming arrows are ordered from left to right, and so are the in-boxes. Similarly, both the outgoing arrows and outboxes are ordered from left to right respectively, because the x-coordinate increases from left to right on the Macintosh screen.



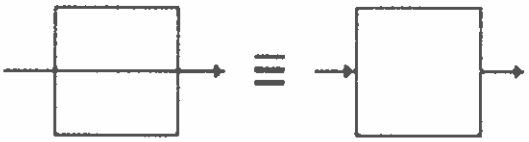
(a) MSTL Binding Rule



(b) Positional Binding Rule



(c) MSTL Convention 1



(d) MSTL Convention 2

Figure 6: MSTL Binding Rules

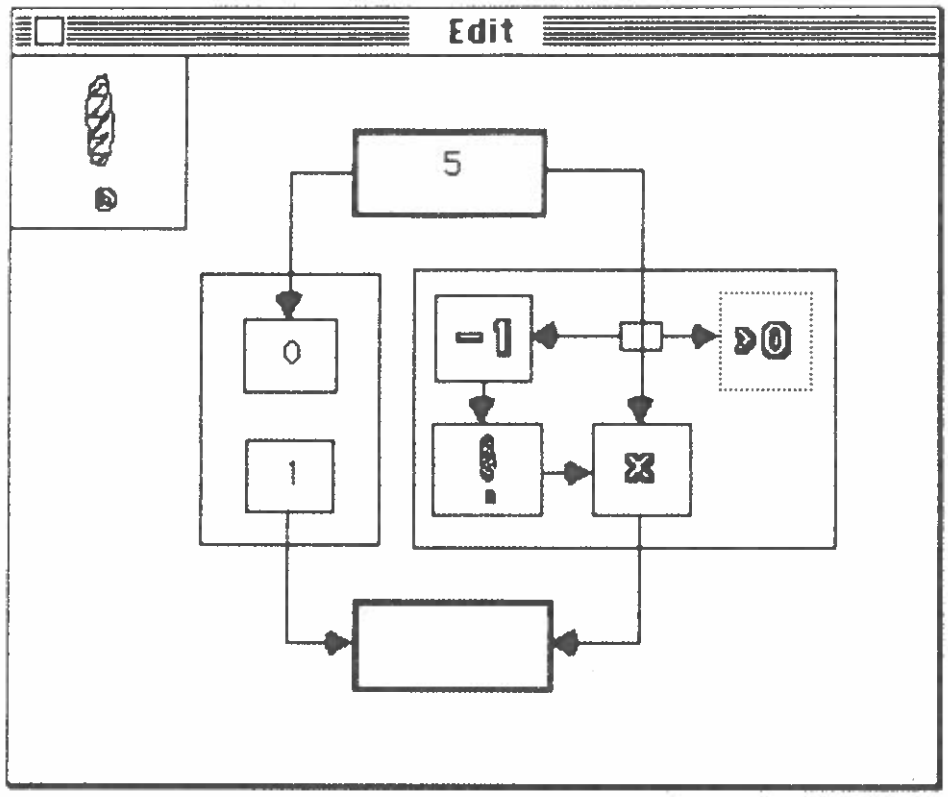
In (b), however, the effect of binding is not obvious as in (a). The four in-boxes {b1, b2, b3, b4} are ordered based on the lexicographical ordering of the points {p1, p2, p3, p4}, and the four arrows {a1, a2, a3, a4} are ordered based on the lexicographical ordering of the points {p5, p6, p7, p8}. Thus, arrows a1, a2, a3, and a4 are bound to boxes b1, b2, b3, and b4, respectively. Figure 6 (c) presents an MSTL convention that allows an arrow to cross the boundary of a complex box in order to make a binding explicit. Figure 6 (d) introduces another MSTL convention for an arrow to intersect with an empty box.

In summary, we will illustrate the notions introduced in this section by a sequence of examples of MSTL programs. We will present, first, a recursive program for the factorial function in Figure 7. Note that in MSTL a nesting of complex boxes such as '!' can be dynamically unfolded as required during the course of computation, and the activation of a complex box can be displayed in a separate window. Thick empty boxes represent input and output parameters.

Next, in Figure 8, we will construct a 4 bit full-adder without using any arithmetic operations, starting with a set of Boolean operations, then a half-adder for binary addition.

Figure 8 (a) through (h) except (b) define a standard set of Boolean operations. The name icons are chosen from the standard symbols for corresponding gate circuits. The boolean values are represented by integers 1 and 0, even though T and F will do as well. Figure 8 (b) is the result of solving (a) with the inputs 1 and 0, and demonstrates

how **and** operation works. Note that the inconsistent box becomes nonextant and nothing flows out of the box. Figure 8 (i) defines a half-adder using the previously defined functions (subroutines), **and** and **exclusive-or**. It has three inputs x, y, and c (carry) from the left, and two outputs, c (next carry) and s (sum). Crossing arrows have no effects on dataflow. Figure 8 (j) is a construction of a full-adder of length 4 using the half-adder of (i), and for testing the construction, the input value (0101) is given for x and (0111) for y, to generate the output (1100) for the sum.



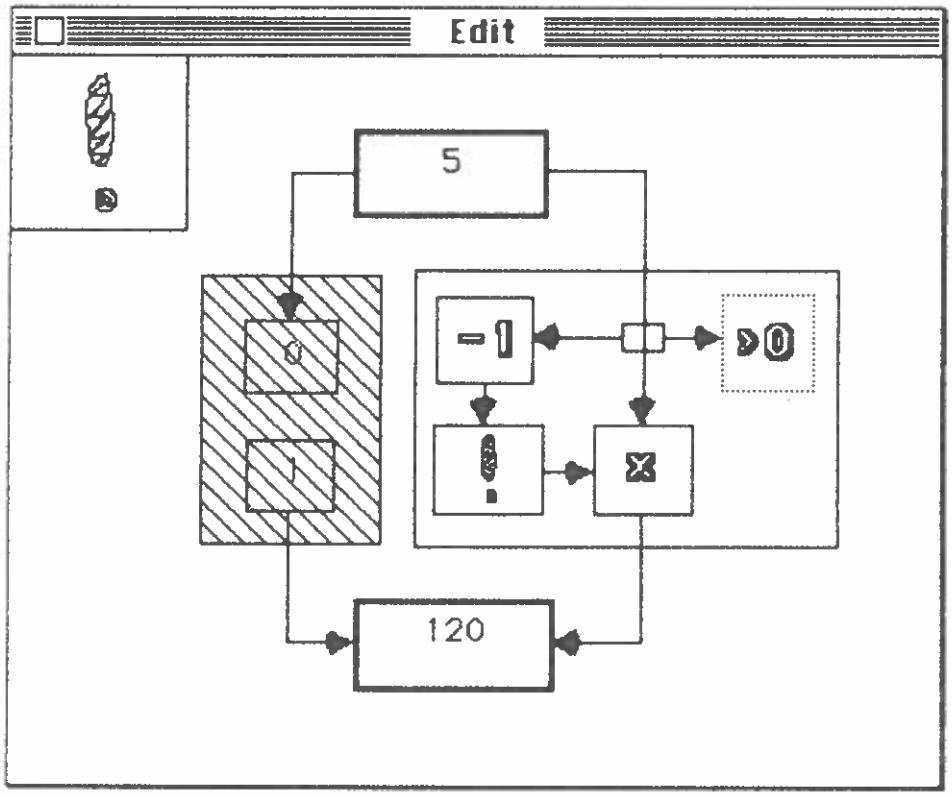


Figure 7: Recursive Definition of Factorial

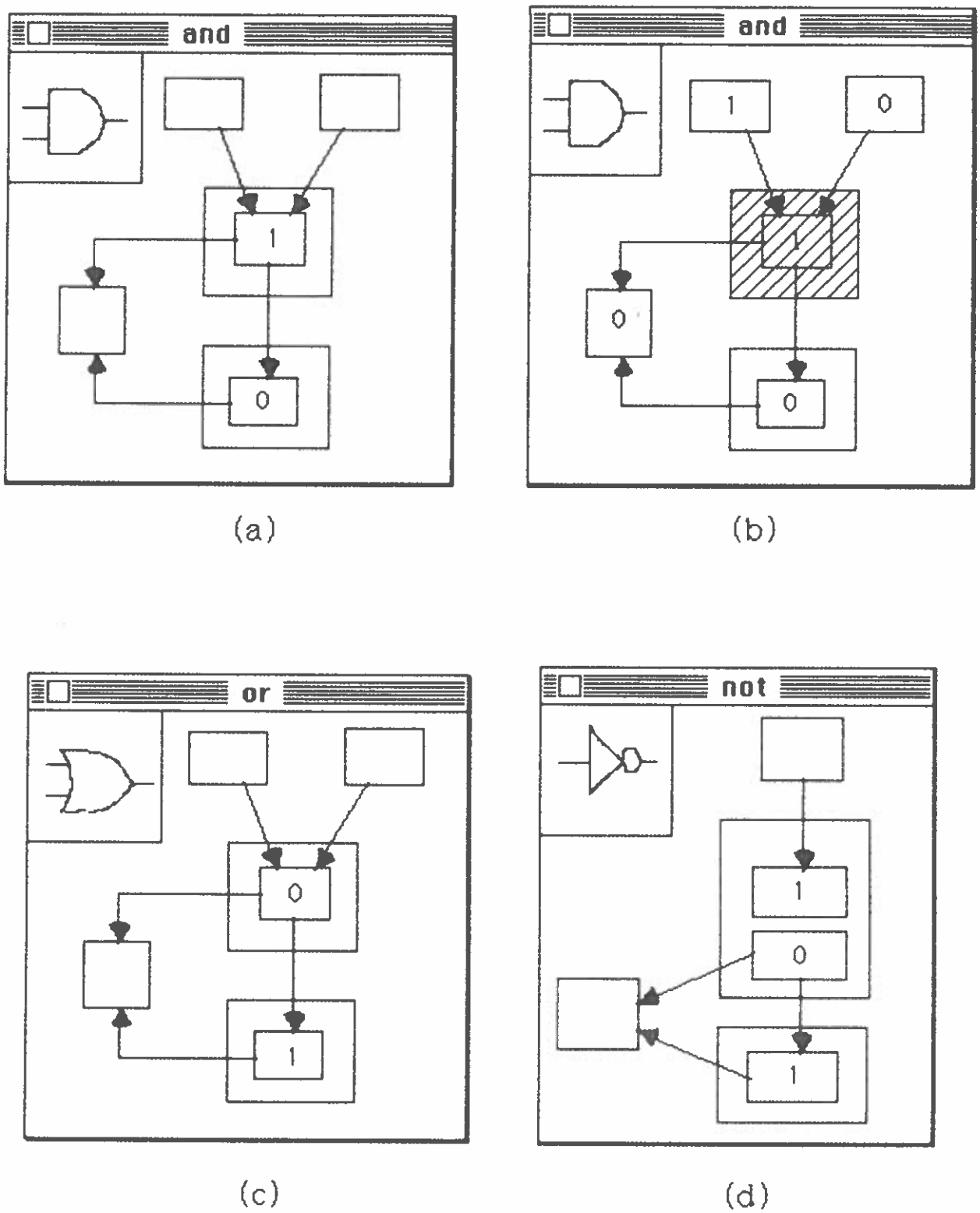
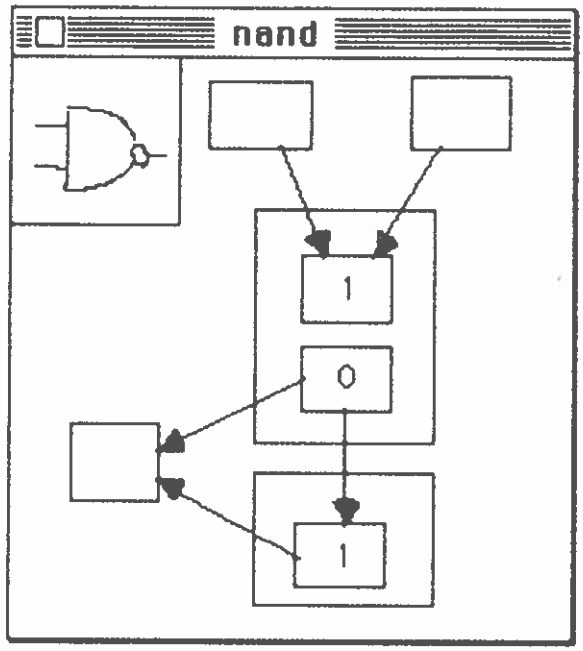
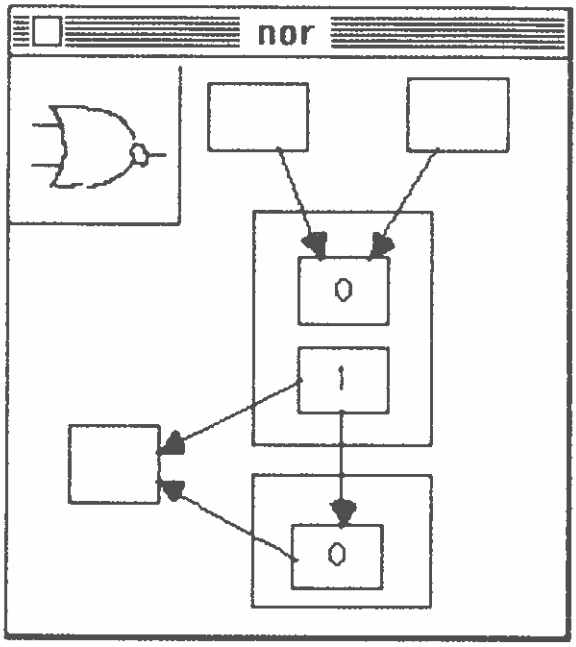


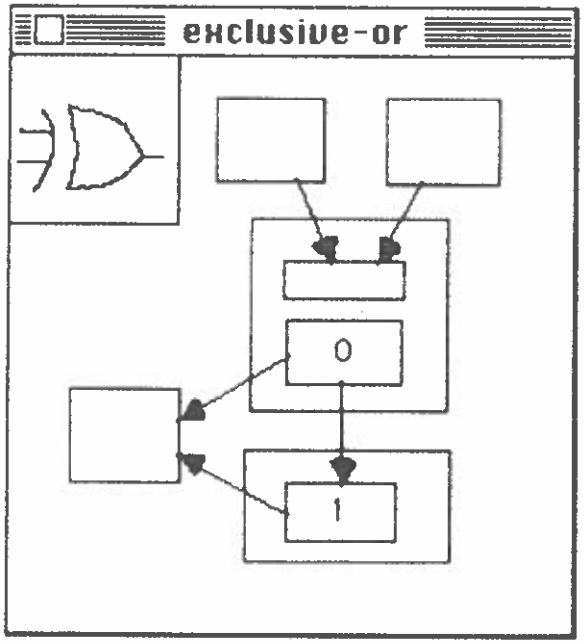
Figure 8: Full-Adder Example



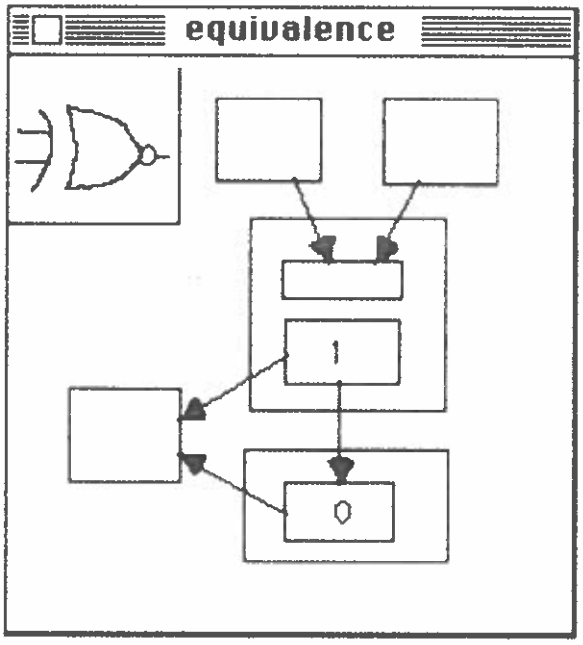
(e)



(f)

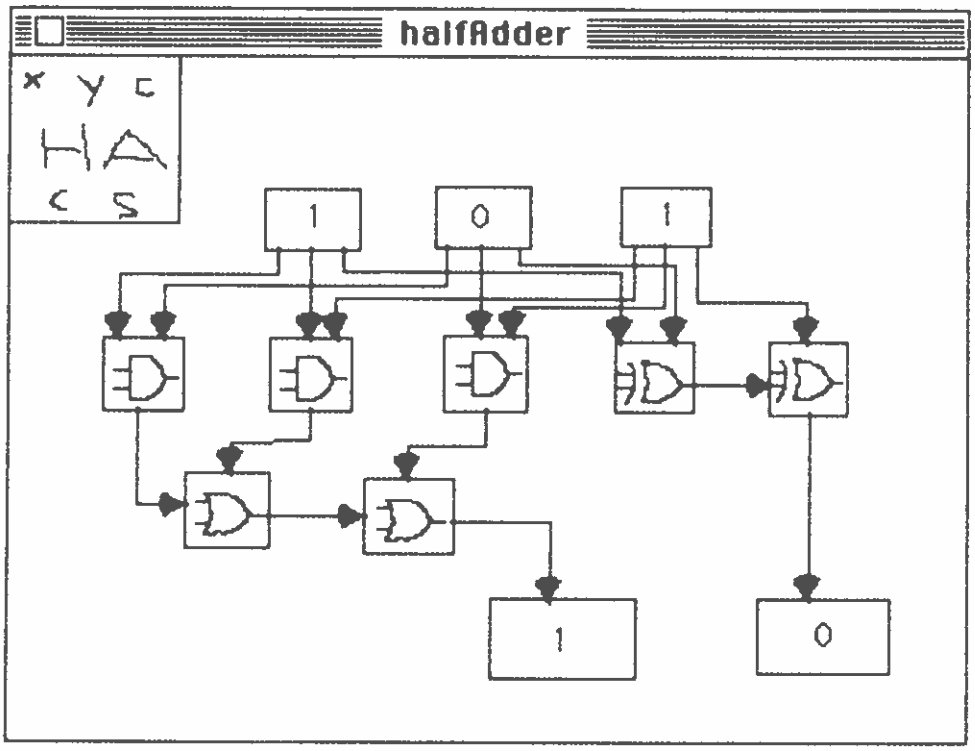


(g)

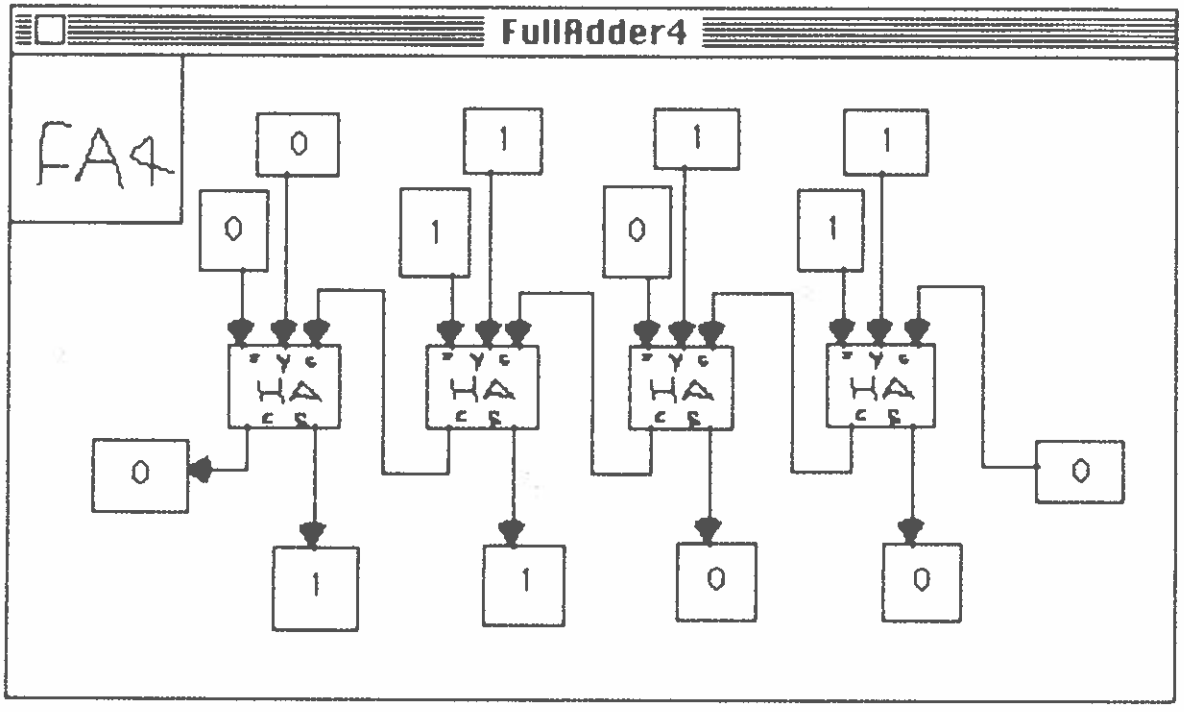


(h)

Figure 8: Full-Adder Example



(i)



(j)

Figure 8: Full-Adder Example

3. Definitions

In this section we will give a formal definition of HDM. The main purpose of formalization is to prove that as a computation model HDM is determinate, i.e., the outcome of a computation specified by a box-graph is unique and independent of the order in which the computation is carried out. The determinancy will be shown in the next section.

There are at least two different ways of defining the semantics of arrows in a box-graph; one is imperative (or prescriptive) and the other is declarative (or descriptive). The imperative interpretation of an arrow connecting box b_1 to box b_2 is : "Move the data in b_1 into b_2." Traditional dataflow models of computation assume this interpretation. The declarative interpretation of the same arrow is : "If b_1 is not empty, then b_2 contains the same data as b_1." This describes the goal state of the action designated by the imperative interpretation of the arrow.

When b_1 and b_2 contains a different data value, there exists a pragmatic difference between the two interpretations. Based on the imperative interpretation, either the content of b_2 will be replaced by the content of b_1 or the operation of data movement will be suspended until b_2 becomes empty. In either way, it is necessary to assume that the box content can be changed from one value to another during the computation.

With the declarative interpretation, on the other hand, an arrow connecting two different values can be considered as an inconsistent statement designating a non-achievable goal. In order to make HDM an applicative model, we will adopt the declarative semantics. Thus, once a box

is filled with a data value, the content of a box never changes. In the following description of HDM, we will add an imperative interpretation as a comment, when it is appropriate to do so.

The following mathematical notations will be used in the remaining part of the report.

$\subset$: is a subset of, (instead of 'is a proper subset of'),

Φ: the empty set,

$\equiv$: equal by definition,

$\Rightarrow$: implies by definition,

$\Leftrightarrow$: if and only if by definition,

$\#(S)$: the cardinarity of set S,

N: the set of natural numbers, {0, 1, 2, ... }.

A *hierarchical dataflow model,* HDM $\equiv$ (T, **G**, τ), consists of a *data type* T, a set of *box-graphs* **G**, and an *evaluation function* τ. The data type defines the domain of computation and the set of primitive operations available on the domain. A box-graph is a two dimensional pictorial expression consisting of arrows and nested boxes. The box-graph defines a logical relationship among the box contents. A box-graph is *consistent* if the content of each box in the box-graph satisfies the constraints imposed by the neighboring box contents. Otherwise *it is inconsistent.* The evaluation function decides whether a box-graph is consistent or not. A *completion problem* is to fill in the empty boxes with data values in such a way that the consistency of the box-graph may be preserved. A *solution* is a set of data values used for the completion.

3.1 Data Type

We define a data type T as a set of functions and predicates defined on the data value set.

$T \equiv (D, F, P)$, where

D is the set of *data objects*,

$F \subset \bigcup_{m,n>0} (D^m \to D^n)$ is the set of *functions* on D,

$P \subset \bigcup_{m>0} (D^m \to 2)$ is the set of *predicates* on D.

The *arity* of a function (or a predicate) is defined as follows:

$\text{arity} : (F \cup P) \to N \times N$

$\text{arity}(f) \equiv (m, n)$ if $f \in D^m \to D^n$,

$\text{arity}(p) \equiv (m, 0)$ if $p \in D^m \to 2$.

Note that any function or a predicate requires at least one argument.

For the semantic definition of box-graph later, we extend the domain D into a complete lattice as follows:

$D^* \equiv D \cup \{\bot, \top\}, \quad \bot, \top \notin D,$

where $\top$ and $\bot$ are the *identity* and the *zero* element of D^*, respectively. The partial ordering is depicted by Figure 9 and defined as: $\bot \leqq d \leqq \top$ for each $d \in D$. We denote the *join* of a set of elements by $\cup$, and the *meet* of a set of elements by $\cap$. Note that by definition $\cup \Phi \equiv \bot$ and $\cap \Phi \equiv \top$. When they are used as infix operators, they denote the operations defined on two elements. Thus,

$d \cup \top = \top$, $d \cap \bot = \bot$, $d \cup d = d$, $d \cap d = d$, $d \cup d' = \top$, $d \cap d' = \bot$,

where $d, d' \in D$ and $d \neq d'$.

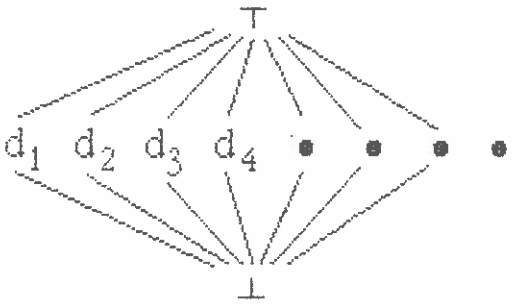


Figure 9 : Partial Ordering on D°

Furthermore,

$D^+ \equiv D \cup \{\perp\} \subset D^\circ$: the set of *data values,*

$\perp$ is the *null value* representing the notion of undefinedness,

$F^+ \equiv \{ f^+ \mid f \in F \}$: the set of naturally extended functions of F, where

$f^+ : (D^+)^m \rightarrow (D^+)^n$ is a *naturally extended* function of f such that

$f^+(x_1, x_2, \ldots, x_m) \equiv (\perp, \ldots, \perp)$, if $x_i = \perp$ for some $1 \leq i \leq m$,

$\equiv f(x_1, x_2, \ldots, x_m)$ otherwise.

$P^+ \equiv \{ p^+ \mid p \in P \}$: the set of naturally extended predicates of P, where

$p^+ : (D^+)^m \rightarrow 2$ is an *extended predicate* of p such that

$p^+(x_1, x_2, \ldots, x_m) \equiv 0$, if $x_i = \perp$ for some $1 \leq i \leq m$,

$\equiv p(x_1, x_2, \ldots, x_m)$ otherwise.

Note that f^+ and p^+ are monotonic functions.

3.2 Box-Graph (The Syntax of HDM)

A box-graph $G \equiv (B, B_o, A, s, d, \mu) \in \mathbf{G}$ is a 6-tuple such that

B is the non-empty finite set of *boxes*,

$B_o \subset B$ is the set of *open* boxes,

($B_c \equiv B - B_o$ is the set of *closed* boxes,)

A is the finite set of *arrows*,

$s : A \rightarrow B$ is the *source* of an arrow,

$d : A \rightarrow B$ is the *destination* of an arrow,

$\mu : B \rightarrow D^+ \cup F^+ \cup P^+ \cup \mathbf{G} \cup \{\phi\}$ is the *content* of a box,

satisfying the conditions (B1) through (B4) given below.

Note that 's' and 'd' are total functions, and therefore each arrow must have the unique starting (source) box and the unique ending (destination) box. A box may contain nothing, a data value, a function, a predicate, or another box-graph. The following terminologies will be used in describing the conditions for the box-graph.

Definition. $b \in B$ is *empty* if $\mu(b) = \phi$.

$base(G) \equiv \{ b \in B \mid \mu(b) = \phi \}$: the *base boxes* of $G \in \mathbf{G}$.

$b \in B$ is *complex* if $\mu(b) \in \mathbf{G}$.

$d^{-1}(b)$: the *incoming arrows* of $b \in B$.

$s^{-1}(b)$: the *outgoing arrows* of $b \in B$.

$pred(b) \equiv s(d^{-1}(b))$: the *predecessor boxes* of $b \in B$.

$succ(b) \equiv d(s^{-1}(b))$: the *successor boxes* of $b \in B$.

$in\text{-}box(G) \equiv \{ b \in B \mid \mu(b) = \phi \wedge d^{-1}(b) = \Phi \}$:
the *in-boxes* of $G \in \mathbf{G}$.

$out\text{-}box(G) \equiv \{ b \in B \mid \mu(b) = \phi \wedge s^{-1}(b) = \Phi \}$:
the *out-boxes* of $G \in \mathbf{G}$.

$arity(G) \equiv (\#(in\text{-}box(G)), \#(out\text{-}box(G)))$:
the *arity* of $G \in \mathbf{G}$.

An empty box is called an in-box if it has no incoming arrow. Similarly, an empty box is called an out-box if it has no outgoing arrow. The in-boxes

and out-boxes of G represent the input parameters and output parameters, respectively, when G is considered to represent an operation. The arity of a box-graph, therefore, is defined as the ordered pair of the number of in-boxes and the number of out-boxes in the box-graph.

With these notions and terminologies we are ready to present the conditions for the box-graph.

Conditions for the box-graph:

(B1) No two boxes intersect with each other.

This is a geometric condition rather than a logical condition. In order to eliminate a semantic ambiguity caused by overlapping boxes, all boxes must be properly nested. Figure10 illustrates an example of such ambiguity. Depending on whether we consider (i) box A contains '+' and box B is empty, (ii) box A is empty and box B contains '+', or (iii) both box A and box B contains '+', the answer could be 4 for (i), 6 for (ii), or 8 for (iii).

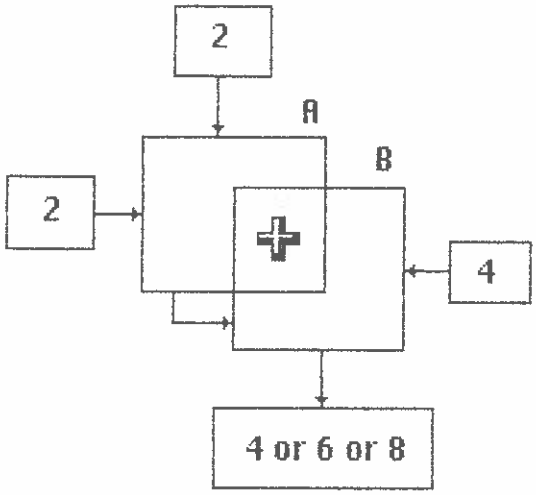


Figure 10: Ambiguous Box-Graph

(B2) (B, A, s, d) is a directed acyclic multi-graph (damg), i.e.,

$(\forall \alpha \in A^*)((\forall i < |\alpha|)(s(\alpha_{i+1}) = d(\alpha_i)) \Rightarrow s(\alpha_1) \neq d(\alpha_{|\alpha|}))$,

where α is a sequence of arrows,

$|\alpha|$ is the length of the sequence,

α_i is the i-th element of the sequence, $0 < i \leq |\alpha|$

There are two main reasons for restricting a box-graph to be acyclic.

First, the acyclicness contributes to making HDM an applicative (functional) model, because once an empty box is filled with a data value, the content will never change in the remaining part of the computation. In other words, HDM is a history independent computation model. As a consequence, an implementation of HDM is not required to keep the entire history (or the state sequence) of computation. We expect that a program derived from HDM is easier to understand because there is no internal state to remember which may affect the execution of the program.

Secondly, the acyclicness makes HDM a deadlock-free and race-free parallel computation model. As we will show in the next section, the model is also determinate and retains the desirable Church-Rosser property.

Note that since the box-graph is a multi-graph, there may be more than one arrow between a pair of boxes. Also note that since the graph is directed and acyclic, (B, <) is a partially ordered set with the following ordering:

$b_1 < b_2 \iff (\exists \alpha \in A^*)$

$(s(\alpha_1) = b_1 \wedge d(\alpha_{|\alpha|}) = b_2 \wedge (\forall i < |\alpha|)(s(\alpha_{i+1}) = d(\alpha_i))$,

i.e., there exists a path from box b_1 to box b_2.

Furthermore, B is a well-founded set because B is finite. These facts will be used later in proving the determinancy of the model by the principle of

structural induction. A box, $b \in B$, is called a *minimum*, if $d^{-1}(b) = \Phi$, because for no $b' \in B$, $b' < b$. Similarly, b is a *maximum*, if $s^{-1}(b) = \Phi$. According to this definition, an in-box is an empty minimum and an out-box is an empty maximum.

(B3) For each box $b \in B$ containing a function, a predicate or another box-graph,

(B3.1) the set of incoming arrows is a totally ordered set and so is the set of outgoing arrows, i.e.,

$d^{-1}(b)$ and $s^{-1}(b)$ are totally ordered sets.

(B3.2) the number of incoming and outgoing arrows of b matches with the arity of its content $\mu(b)$, i.e.,

$$(\forall b \in B)(\ \mu(b) \in F^+ \cup P^+ \cup G \Rightarrow$$
$$(\ \#(d^{-1}(b)), \#(s^{-1}(b))\ ) = \text{arity}(\mu(b))\ ).$$

The condition (B3.1), along with the next (B4), will be used to associate the contents of the predecessors and successors of a box with the content of the box. They define the binding rule for the operation represented by the content of the box. Note that since $d^{-1}(b)$ and $s^{-1}(b)$ are totally ordered, so are pred(b) and succ(b). In MSTL, the ordering of $d^{-1}(b)$ is defined geometrically by the lexicographical ordering of the (X, Y)-coordinates of arrow heads residing on the b's boundary, and the ordering of $s^{-1}(b)$ is based on the (X, Y)-coordinates of the tails of arrows on b.

The condition (B3.2) is necessary for preserving semantic integrity of data type operations. Note that this condition applies to a complex box where $\mu(b) \in G$, $\text{arity}(\mu(b)) \equiv (\#(\text{in-box}(\mu(b))), \#(\text{out-box}(\mu(b))))$.

However, the empty boxes and the boxes that contains a data value need not satisfy this condition because they do not represent an operation.

(B4) For each box-graph G $\in$ **G**, the in-boxes of G is a totally ordered set, and so is the out-boxes of G.

This condition is used to specify the binding rule for complex boxes. In MSTL, the ordering of boxes is also defined geometrically by the lexicographical ordering of the (X, Y)-coordinate of the left-upper corner of each box. It is a consequence of conditions (B3) and (B4) that for every complex box b, there exists an one-to-one correspondence between the incoming arrows of b and the in-boxes of μ(b), and between the outgoing arrows of b and the out-boxes of μ(b). Figure 6 (a) of Page12 illustrates the binding rule in MSTL.

(End of conditions of *box-graph*)

3.3 Evaluation Function (The Semantics of HDM)

The most fundamental property of a box-graph is whether it is consistent or not, i.e., whether the content of a box satisfies the local constraints imposed by the neighboring boxes. The evaluation function τ is defined as a characteristic function for consistent graphs.

In order to define the notion of consistency more precisely, we introduce the semantic concept of elaboration, which is an assignment of data values to the set of arrows. The concept of elaboration in HDM is similar to the concept of interpretation in mathematical logic. A box-graph is consistent if there exists an elaboration for the box-graph, as, in mathematical logic, a set

of propositions are consistent if there exists a model (interpretation) for the set. The semantics of HDM is declarative rather than imperative. The consistency of a box-graph is dependent on the existence of an elaboration and is independent of how the elaboration is constructed.

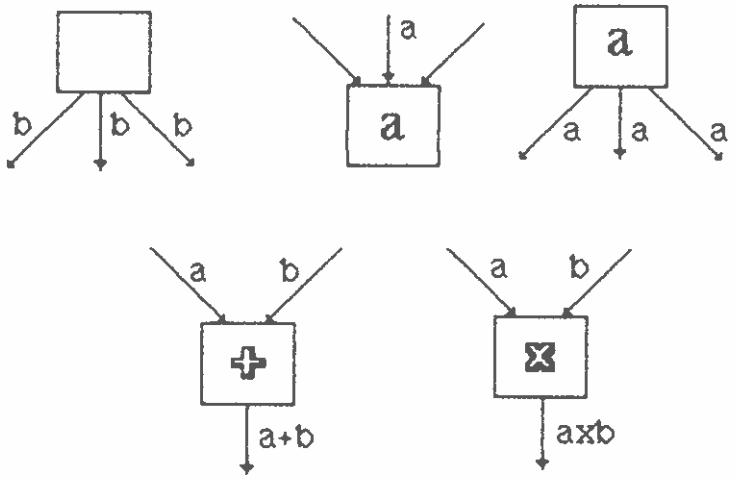
Given an elaboration of a box-graph, the data values assigned to the incoming and outgoing arrows on an individual box should satisfy the local conditions imposed by the content of the box. Figure11 illustrates the concept of elaboration by a simple example. Figure11 (a) specifies the local constraints imposed by various boxes occurring in (b) and (c). An empty box imposes no constraint except that all outgoing arrows must have the same value. A box that contains a data value must have the same value as the content of the box for all outgoing arrows and for all non-null incoming arrows. The formal definition of these constraints will be given later in this section. The box-graph of (b) is consistent because it has an elaboration whose values are assigned to each corresponding arrow in the box-graph. For the box-graph of (c) there is no such an elaboration, therefore the box-graph is inconsistent.

A mapping $\theta : A \rightarrow D^+$ is an *elaboration* of $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$, if the following conditions (E1) through (E5) hold for any $b \in B$.

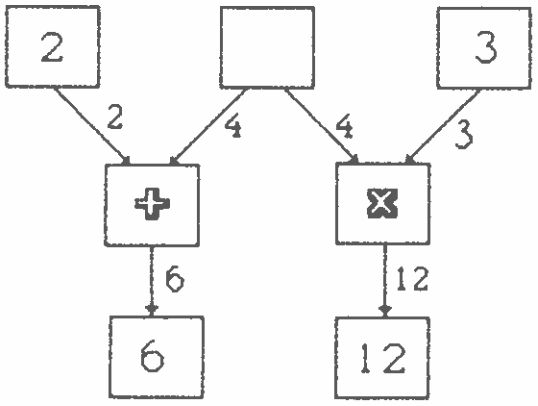
Notation: We will denote the i-th in-arrow of b by a_i, and the j-th out-arrow by A_j, where $1 \leq i \leq m$, and $1 \leq j \leq n$. (See Figure 12.) Note that $d^{-1}(b)$ and $s^{-1}(b)$ are totally ordered sets by the condition (B3.1), and they can be represented as sequences:

$$d^{-1}(b) \equiv (a_i) \equiv (a_1, a_2, \ldots, a_m),$$

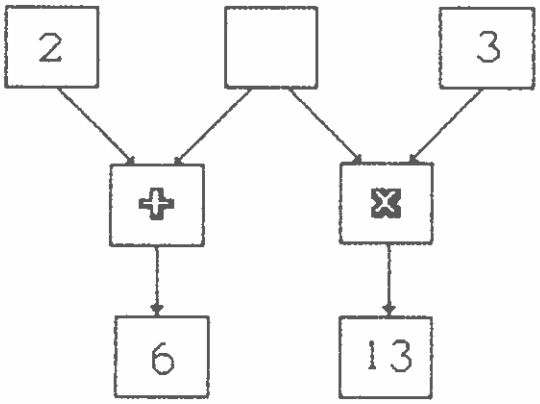
$$s^{-1}(b) \equiv (A_j) \equiv (A_1, A_2, \ldots, A_n).$$ (End of Notation)



(a) Local Constraints



(b) Consistent Box-Graph



(c) Inconsistent Box-Graph

Figure 11: Elaboration of Box-Graph

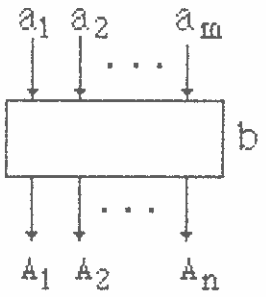


Figure 12: In-Arrows and Out-Arrows

Conditions for the Elaboration:

Let θ be an elaboration of $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$, then for any $b \in B$:

(E1) When b is empty, i.e., $\mu(b) = \phi$.

There are three sub-conditions; on the in-arrows, on the out-arrows, and on the relationship between them.

(E1.1) $\bigcup \{\theta(a_i) \mid a_i \in d^{-1}(b)\} \in D^+$.

No two in-arrows have distinct values of D under θ.

(E1.2) $(\forall A,A' \in s^{-1}(b))(\theta(A)=\theta(A'))$ where $s^{-1}(b) \neq \Phi$.

Every out-arrow has the same value of D^+ under θ.

(E1.3) $\theta(A) = \bigcup \{\theta(a_i) \mid a_i \in d^{-1}(b)\}$ where $d^{-1}(b) \neq \Phi \wedge A \in s^{-1}(b)$.

Each out-arrow has the least upper bound (lub) of the values assigned to the in-arrows under θ.

If $d^{-1}(b) = \Phi$, then (E1.3) does not apply and any value can be assigned to the out-arrows as long as they are all the same. If $s^{-1}(b) = \Phi$, then only (E1.1) applies. Note that when $d^{-1}(b) \neq \Phi$ the values of the out-arrows are the same member of D^+ and uniquely determined by the values of the in-arrows. Whether b is a closed box or an open box does not matter.

An empty box may receive data from any number of boxes provided that they are the same. It sends the received datum to the successors. If it receives different data, the box-graph G must be inconsistent.

(E2) When b contains a data value, i.e., $\mu(b) \equiv c \in D^+$.

(E2.1) $\bigcup \{\theta(a_i) \mid a_i \in d^{-1}(b)\} \leqq c$.

If an in-arrow has a value of D, then it must be $\mu(b)$.

(E2.2) $(\forall A \in s^{-1}(b))(\theta(A) = c)$ where $s^{-1}(b) \neq \Phi$.

Each out-arrow has $\mu(b)$ as its value under θ.

If $s^{-1}(b) = \Phi$, then only (E2.1) applies. Note that the in-arrows may have the null value, but the out-arrows must have the same value as $\mu(b) \in D$. Whether b is a closed box or an open box does not matter.

A data box may receive values from any number of boxes provided that the value is identical to its content. It sends its content to all of the successor boxes. If the data box receives a different value from its content, then the box-graph G must be inconsistent.

(E3) When b contains a function f, i.e., $\mu(b) \equiv f^+ \in F^+$, where

$$f^+ : D^m \rightarrow D^n \text{ for some } m,n > 0.$$

Note that the arity of b is the same as the arity of $\mu(b)$ by the condition (B3.2) for the box-graph and that the $s^{-1}(b)$ and $d^{-1}(b)$ are totally ordered sets by the condition (B3.1). There are two cases depending on whether b is a closed box or not.

Case1: When b is a closed box, i.e., $b \in B_c$.

The values on the out-arrows are the values of the function f^+ evaluated with the values on the in-arrows as the arguments. If some

in-arrow has the null value, then all of the out-arrows must have the null value, i.e.,

(E3.1) $\Theta(s^{-1}(b)) \equiv f^{+}(\Theta(d^{-1}(b)))$, or equivalently,
$(\Theta(A_1), \Theta(A_2), \ldots, \Theta(A_n)) = f^{+}(\Theta(a_1), \Theta(a_2), \ldots, \Theta(a_m))$.

Case 2: When b is an open box, i.e., $b \in B_0$.

All of the values for the in-arrows are defined and the condition is the same as the Case 1. Thus,

(E3.2) $\Theta(d^{-1}(b)) \in D^m \wedge \Theta(s^{-1}(b)) = f^{+}(\Theta(d^{-1}(b)))$.

A function box receives all arguments from the predecessor boxes, and sends results to the successor boxes. Since the functions and predicates in HDM are considered as system defined box-graphs, they are either consistent or inconsistent. If not all input arguments are defined, then the function itself, $\mu(b)$, instead of G, must be considered inconsistent. Thus, if b is a closed box, then the out-arrows must have the null value. If b is an open box, then both $\mu(b)$ and G must be considered inconsistent.

(E4) When b contains a predicate p, i.e., $\mu(b) \equiv p^{+} \in P^{+}$, where

$p^{+} : (D^{+})^m \rightarrow 2$ for some $m > 0$.

Note that $s^{-1}(b) = \Phi$ by the condition (B3.2). If b is a closed box, then there is no condition on Θ with respect to b. If b is an open box, then all of the values for the in-arrows must satisfy the predicate p^{+}, i.e.,

$p^{+}(\Theta(d^{-1}(b)))$.

Note that $p^{+}(\Theta(d^{-1}(b))) = 0$ if $\Theta(a) = \perp$ for some $a \in d^{-1}(b)$.

A predicate box receives all arguments from the predecessor boxes, and tests whether they satisfy the condition specified by the predicate. If the

condition is satisfied, then G is consistent regardless whether the box is open or closed. If the condition is not satisfied and, furthermore, the box is open, then G must be inconsistent. If some input arguments are undefined when the box is open, then G as well as the predicate must be considered as inconsistent.

(E5) When b contains another box-graph, i.e., $\mu(b) \in \mathbf{G}$, where

$$\mu(b) \equiv \mathbf{G} \equiv (\mathbf{B}, \mathbf{B}_0, \mathbf{A}, \mathbf{s}, \mathbf{d}, \mu) \in \mathbf{G}.$$

Note that in-box($\mathbf{G}$) and out-box($\mathbf{G}$) are totally ordered sets by the condition (B4) of §3.2. Similarly, $d^{-1}(b)$ and $s^{-1}(b)$ are totally ordered sets by the condition (B3.1) of §3.2. By the condition (B3.2), there is an one-to-one correspondence between in-box($\mathbf{G}$) and $d^{-1}(b)$ as follows:

$$\text{in-box}(\mathbf{G}) \equiv (\mathbf{b}_i) \equiv (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m),$$

$$d^{-1}(b) \equiv (a_i) \equiv (a_1, a_2, \ldots, a_m).$$

Similarly there is an one-to-one correspondence between out-box($\mathbf{G}$) and $s^{-1}(b)$ as follows:

$$\text{out-box}(\mathbf{G}) \equiv (\mathbf{B}_j) \equiv (\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n),$$

$$s^{-1}(b) \equiv (A_j) \equiv (A_1, A_2, \ldots, A_n).$$

Note also that by definition $\mathbf{b}_i$ does not have any incoming arrow and $\mathbf{B}_j$ does not have any outgoing arrow in $\mathbf{G}$. See Figure 13 below.

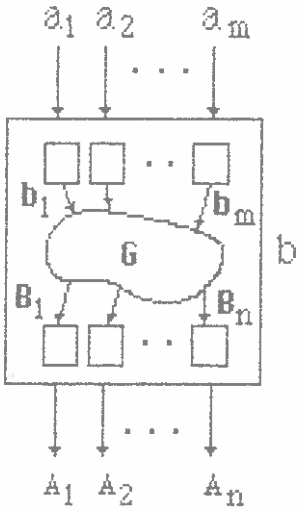


Figure 13: Binding Rule

There are two cases depending on whether b is a closed box or not. The following notations for the update function will be used in presenting the two conditions:

Notations: Let $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$.

(a) $G/\mu' \equiv (B, B_0, A, s, d, \mu')$ for any content function μ'.

(b) $f[x/y]$ is the *update* function of f, i.e.,

$$\begin{cases} f[x/y](z) \equiv x & \text{if } y = z, \\ \equiv f(z) & \text{otherwise.} \end{cases}$$

(c) $f[x_i/y_i \mid y_i \in S]$ is the *multiple update* function of f, i.e.,

$$\begin{cases} f[x_i/y_i \mid y_i \in S](z) \equiv x_i & \text{if } z = y_i \in S, \\ \equiv f(z) & \text{otherwise.} \end{cases}$$

(End of Notations)

Case 1: When b is a closed box, i.e., $b \in B_c$.

Either **G** has an elaboration Θ that satisfies the binding conditions imposed by $\Theta(d^{-1}(b))$ and $\Theta(s^{-1}(b))$, or all the out-arrows of b must have the null value under Θ. More precisely:

(E5.1) Let $\mathbf{G}' \equiv \mathbf{G}/\mu[\Theta(a_i)/\mathbf{b}_i \mid \mathbf{b}_i \in \text{in-box}(\mathbf{G})]$.

If $\mathbf{G}'$ has an elaboration Θ

then $\Theta(A_j) = U\{\Theta(\mathbf{a}) \mid \mathbf{a} \in \mathbf{d}^{-1}(\mathbf{B}_j)\}$ for every $\mathbf{B}_j \in \text{out-box}(\mathbf{G})$

else $\Theta(A_j) = \perp$ for every $\mathbf{B}_j \in \text{out-box}(\mathbf{G})$.

A closed box containing a box-graph **G** receives input values (input parameters) from the predecessor boxes, moves them into the corresponding in-boxes of **G**, and solves (completes) it. If it is successfully solved, the contents of the out-boxes of **G** will be moved to the successor boxes. Otherwise, no values will be moved to the successor boxes.

Case 2: When b is an open box, i.e., $b \in B_o$.

G has an elaboration Θ that satisfies the binding conditions imposed by $\Theta(d^{-1}(b))$ and $\Theta(s^{-1}(b))$, i.e.,

(E5.2) $\mathbf{G}'$ has an elaboration Θ $\wedge$

$\Theta(A_j) = U\{\Theta(\mathbf{a}) \mid \mathbf{a} \in \mathbf{d}^{-1}(\mathbf{B}_j)\}$ for every $\mathbf{B}_j \in \text{out-box}(\mathbf{G})$.

An open box containing a box-graph **G** behaves exactly the same as a closed box containing the same box-graph, except that when the in-boxes of **G** are filled with the contents of the predecessor boxes, and **G** fails to complete, the entire box-graph G that contains the open box is to be considered inconsistent. (End of Conditions for Elaboration)

A box-graph $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$ is *consistent* if there exists an elaboration for G, otherwise it is *inconsistent*.

The evaluation function $\tau : \mathbf{G} \rightarrow 2$ is defined as :

$$\begin{cases} \tau(G) \equiv 1 & \text{if G is consistent,} \\ \equiv 0 & \text{otherwise.} \end{cases}$$

(End of definition of *Evaluation Function*)

3.4 Solution

We will define the notion of solution for an arbitrary consistent box-graph G as a set of values that completes G. Formally an assignment σ of data values to the base of G is called a *solution* for G if the completed box-graph G_σ is consistent, where G_σ is the same as G except that its base boxes are filled with the data values assigned by σ. That is,

$\sigma : \text{base}(G) \rightarrow D^+$ is a *solution* for G $\langle = \rangle$ $\tau(G_\sigma) = 1$,

where

$G_\sigma \equiv G/\mu[\sigma(b)/b \mid b \in \text{base}(G)]$.

The following theorem establishes the relationship between the notion of elaboration and the notion of solution. The theorem shows that a solution for G can be constructed directly from an elaboration of G and the construction is unique.

Theorem 1:

Let $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$, and let Θ be an elaboration of G.

Then, $\sigma_\Theta : \text{base}(G) \rightarrow D^+$ is a solution for G,

where $\sigma_\Theta(b) \equiv \cup (\Theta(a_i), \Theta(A_j) \mid a_i \in d^{-1}(b), A_j \in s^{-1}(b))$.

Proof: We want to show that $G/\mu[\sigma_\theta(b)/b \mid b \in base(G)]$ is consistent. We claim that θ is an elaboration of $G/\mu[\sigma_\theta(b)/b]$ for any $b \in base(G)$.

Let $v \equiv \sigma_\theta(b)$. We want to show that if θ satisfies the conditions (E1) for the box b in G, then θ also satisfies the conditions (E2) for b in $G' \equiv G/\mu[v/b]$.

Assume that θ satisfies (E1). Then,

$v = \sigma_\theta(b) \equiv \cup\{\theta(a_i), \theta(A_j) \mid a_i \in d^{-1}(b),\ A_j \in s^{-1}(b)\}$

$= (\cup\{\theta(a_i) \mid a_i \in d^{-1}(b)\}) \cup (\cup\{\theta(A_j) \mid A_j \in s^{-1}(b)\})$

by the definition of $\cup$,

$= (\cup\{\theta(a_i) \mid a_i \in d^{-1}(b)\}) \cup \theta(A_j)$ by (E1.2),

$= \cup\{\theta(a_i) \mid a_i \in d^{-1}(b)\}$ by (E1.3).

Since $v = \cup\{\theta(a_i) \mid a_i \in d^{-1}(b)\} \leqq \sigma_\theta(b) = v$, θ satisfies (E2.1) for b in G'. By the condition (E1.3), $\theta(A) = v$, i.e., θ also satisfies (E2.2).

(End of Proof)

Note: σ_θ is called the *minimum solution* for G (derived from θ), because it can be shown below that for any solution σ for G, $\sigma_\theta \leqq \sigma$, i.e., for any $b \in B$, $\sigma_\theta(b) \leqq \sigma(b)$. Let σ be an arbitrary solution of G. Since G has the unique elaboration θ by (E2), for any base box b of G,

$\sigma_\theta(b) \equiv \cup\{\theta(a_i), \theta(A_j) \mid a_i \in d^{-1}(b),\ A_j \in s^{-1}(b)\} \leqq \sigma(b)$.

(End of Note)

A box-graph is called *bound* if every minimum box contains a data value or another bound box-graph, otherwise it is called *free*. In the next section, we will show that if a box-graph is bound and consistent, then the elaboration is unique, therefore the minimum solution is unique. Because of

this determinancy result, a free box-graph can be considered as a specification of a function, or a specification of a predicate when the box-graph has no out-boxes. The notion of bound box-graph will be formally defined in the next section.

4. Determinancy

In the previous sections we defined the concepts of consistency, elaboration and solution of a box-graph. In this section we will show that HDM is determinate, i.e., if a box-graph has a minimum solution, then it is unique. We will also present an algorithm by which the minimum solution can be derived.

4.1 The Uniqueness of Elaboration

The consistency of a box-graph is not sufficient for the uniqueness of a solution for the box-graph. For example, a box-graph consisting of a single empty box is consistent but an assignment of any data value to the empty box is a solution because there is no constraint imposed on the content. However, the minimum solution for the single box box-graph, that is, the assignment of $\perp$ to the box, is unique. We will introduce a sufficient condition for the uniqueness of a minimum solution for a box-graph.

Definition: Let $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$.

G is *bound* if every minimum box of G contains a data value or another bound box-graph, i.e.,

$(\forall b \in B)(d^{-1}(b) = \Phi \Rightarrow (\mu(b) \in D^{+} \vee (\mu(b) \in \mathbf{G} \wedge \mu(b) \text{ is bound })))$.

Otherwise, G is *free*.

We will show that the above condition is sufficient for the uniqueness of an elaboration for a box-graph. By the definition of minimum solution, the uniqueness of elaboration entails that of minimum solution. Note that the condition is not sufficient for the uniqueness of a solution. For example, even though the box-graph of Figure 14 is consistent and bound, any

assignment of any data value to the out-box is a solution. However, there is only one minimum solution for the box-graph, i.e., the assignment of $\perp$.

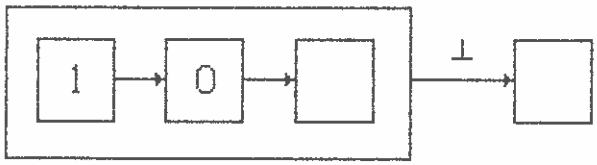


Figure 14: Consistent and Bound Box-Graph

Theorem 2:

Let $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$ be a consistent and bound box-graph.

Then, the minimum solution for G is unique, i.e.,

if $\sigma_1, \sigma_2 : \text{base}(G) \rightarrow D^+$ are minimum solutions of G,

then $\sigma_1(b) = \sigma_2(b)$ for any $b \in \text{base}(G)$.

The main tool for the proof is the principle of *structural induction* for well-founded sets[8]. A partially ordered set (poset) is *well-founded* if there is no infinite sequence of decreasing elements. Every well-ordered set is well-founded. Every finite poset is well-founded. Let **P**(x) be an arbitrary predicate on a well-founded poset (X, $\leqq$). The principle of structural induction states that: To show that $(\forall x \in X)$ **P**(x), it is sufficient to show that $(\forall x \in X)(((\forall y \leqq x)\ \mathbf{P}(y)) \Rightarrow \mathbf{P}(x))$. This condition implies that **P**(x) holds for each minimal element x of X.

Proof of Theorem 2: We will show that the elaboration for a consistent bound box-graph is unique:

(P1) If θ_1 and θ_2 are elaborations of a bound box-graph $G \in \mathbf{G}$,

then $\theta_1 = \theta_2$.

We use two induction principles in this proof; the mathematical induction on the levels of nesting of box-graphs, and the structural induction on the partial ordered set of boxes in a box-graph.

First we define the *level of nesting* λ for a box-graph $G \in \mathbf{G}$ as follows:

$\lambda : \mathbf{G} \to N$ and

$$\begin{cases} \lambda(G) \equiv 0 & \text{if } \mathrm{complex}(G) = \phi, \\ \quad \equiv \max \{ \lambda(\mu(b)) \mid b \in \mathrm{complex}(G) \} + 1 & \text{otherwise,} \end{cases}$$

where $\mathrm{complex}(G) \equiv \{ b \in B \mid \mu(b) \in \mathbf{G} \}$.

(I) Base Step (Mathematical Induction): We will show that (P1) holds when $\lambda(G) = 0$, i.e., when G has no complex boxes.

Note that by the condition (B2) of §3.2, $(B, <)$ is a partially ordered set, and B is well-founded because it is finite. Let θ_1 and θ_2 be elaborations of G. We will show by the principle of structural induction that:

(P2) $\quad (\forall b \in B)\, \mathbf{P}(b)$

where $\mathbf{P}(b) \equiv [(\theta_1(d^{-1}(b)) = \theta_2(d^{-1}(b))) \Rightarrow (\theta_1(s^{-1}(b)) = \theta_2(s^{-1}(b)))]$.

Note that (P2) is a sufficient condition for (P1), because if $\theta_1 \neq \theta_2$, then $\theta_1(a) \neq \theta_2(a)$ for some $a \in s^{-1}(b)$ and for some $b \in B$.

Let $b \in B$ be an arbitrary box in G, and assume that $\mathbf{P}(b')$ for any b' such that $b' < b$. We want to show that $\mathbf{P}(b)$. We divide the argument into two cases; when b has no such b' and when b has such a predecessor b'.

Case 1: When b is a minimum element of B, i.e., $d^{-1}(b) = \Phi$.

We claim that $\mu(b) \in D^+$. Since G is bound and b is a minimum element, $\mu(b) \neq \phi$, i.e., b cannot be empty. Since every function and predicate requires at least one argument by the definition of F^+ and P^+, and $d^{-1}(b) = \Phi$, $\mu(b)$ cannot be a function or a predicate. Furthermore, since $\lambda(G) = 0$, $\mu(b)$

cannot be a box-graph. Therefore, it must be the case that $\mu(b) \in D^+$. By the definition of Elaboration Condition (E2.2),

$$\theta_1(s^{-1}(b)) = \theta_2(s^{-1}(b)) = \mu(b).$$

Therefore, **P**(b) holds when $d^{-1}(b) = \Phi$. (End of Case 1)

Case 2: When $d^{-1}(b) \neq \Phi$.

Assume that $\theta_1(d^{-1}(b)) = \theta_2(d^{-1}(b))$. We want to show that

$$\theta_1(s^{-1}(b)) = \theta_2(s^{-1}(b)),$$

i.e., $\theta_1(a) = \theta_2(a)$ for any $a \in s^{-1}(b)$.

There are four cases depending on $\mu(b)$:

(1) $\mu(b) = \phi$.

By the Elaboration Condition (E1.3), for any $A \in s^{-1}(b)$,

$\theta_1(A) = \cup\{\theta_1(a_i) \mid a_i \in d^{-1}(b)\}$,

$\theta_2(A) = \cup\{\theta_2(a_i) \mid a_i \in d^{-1}(b)\}$.

Since $\theta_1(a_i) = \theta_2(a_i)$ for any $a_i \in d^{-1}(b)$,

$\theta_1(A) = \theta_2(A)$ for any $A \in s^{-1}(b)$.

(2) $\mu(b) \in D^+$. By the definition of Elaboration Condition (E2.2);

$\theta_1(s^{-1}(b)) = \theta_2(s^{-1}(b)) = \mu(b)$.

(3) $\mu(b) \in F^+$.

Since $\theta_1(d^{-1}(b)) = \theta_2(d^{-1}(b))$ by the assumption,

$\theta_1(s^{-1}(b)) = \mu(b)(\theta_1(d^{-1}(b))) = \mu(b)(\theta_2(d^{-1}(b))) = \theta_2(s^{-1}(b))$

by (E3.1) if $b \in B_c$, and by (E3.2) if $b \in B_o$.

(4) $\mu(b) \in P^+$.

Vacuously true, because $s^{-1}(b) = \Phi$ by (B3.2) of §3.2.

By (1) through (4), **P**(b) holds when $d^{-1}(b) \neq \Phi$. (End of Case 2)

From Case1 and Case2, (P2) holds when $\lambda(G) = 0$. (End of Base Step)

(II) Induction Step (Mathematical Induction):

Assume that (P1) holds for any consistent bound box-graph G whose level of nesting is less than $n > 0$, i.e.,
$(\lambda(G) < n) \wedge (G \text{ is bound}) \wedge (\theta_1 \text{ and } \theta_2 \text{ are elaborations of } G) \Rightarrow \theta_1 = \theta_2$.

Let $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$ be a consistent and bound box-graph of the nesting level $n > 0$, and let θ_1 and let θ_2 be elaborations of G. We will show that $\theta_1 = \theta_2$ by showing that $(\forall b \in B)\ \mathbf{P}(b)$. As in (I) Base Step, we divide the argument into two cases; when b does not have a predecessor and when it does.

Case 1: When b is a minimum element of B, i.e., $d^{-1}(b) = \Phi$.

By the same argument as in (I) Base Step Case1, $\mu(b) \in D^+$ or $\mu(b) \in \mathbf{G}$. Also, as in (I) Base Step Case1, $\mathbf{P}(b)$ holds when $\mu(b) \in D^+$. Therefore we only have to show that $\mathbf{P}(b)$ holds when $\mu(b) \in \mathbf{G}$.
Let $\mu(b) \equiv \mathbf{G} \equiv (\mathbf{B}, \mathbf{B_0}, \mathbf{A}, \mathbf{s}, \mathbf{d}, \boldsymbol{\mu}) \in \mathbf{G}$, then by the condition (3.2) of §3.2, in-box($\mathbf{G}$) = Φ, and since G is bound, so is $\mathbf{G}$. Either $\mathbf{G}$ is consistent or inconsistent.

(1) Assume that $\mathbf{G}$ is inconsistent. Then, b must be a closed box, because if b is an open box and $\mathbf{G}$ (= $\mu(b)$) is inconsistent, then G must be inconsistent, contradicting with the definition of G. By the condition (E5.1) of §3.3, $\theta_1(a) = \theta_2(a) = \perp$ for any $a \in s^{-1}(b)$, and $\mathbf{P}(b)$ is true.

(2) Assume that $\mathbf{G}$ is consistent and let $\boldsymbol{\theta}$ be an elaboration of $\mathbf{G}$, then since $\mathbf{G}$ is bound and $\lambda(\mathbf{G}) < n$, by the induction hypothesis, $\boldsymbol{\theta}$ is unique.

Let a_j be the j-th out-arrow of b, where $1 \leq j \leq \#(s^{-1}(b))$. We want to show that $\theta_1(a_j) = \theta_2(a_j)$. Since θ_1 and θ_2 are elaborations of G, they must satisfy the Elaboration Condition (E5.1), i.e.,

$\theta_1(A_j) = \cup\{\Theta(\mathbf{a}) \mid \mathbf{a} \in \mathbf{d}^{-1}(B_j)\}$,

$\theta_2(A_j) = \cup\{\Theta(\mathbf{a}) \mid \mathbf{a} \in \mathbf{d}^{-1}(B_j)\}$.

Therefore, $\theta_1(A_j) = \theta_2(A_j)$, and **P**(b) holds when b is a minimum element of B and $\mu(b) \in \mathbf{G}$. (End of Case 1)

Case 2: When b has a predecessor, i.e., $d^{-1}(b) \neq \Phi$.

Assume that $\theta_1(d^{-1}(b)) = \theta_2(d^{-1}(b))$. We want to show that

$\theta_1(s^{-1}(b)) = \theta_2(s^{-1}(b))$, i.e., for any $A_j \in s^{-1}(b)$, $\theta_1(A_j) = \theta_2(A_j)$.

There are five cases to consider, depending on $\mu(b)$. However, the following first four cases are the same as in (I) Base Step, Case 2.

(1) $\mu(b) = \phi$. (2) $\mu(b) \in D^+$. (3) $\mu(b) \in F^+$. (4) $\mu(b) \in P^+$.

Therefore, we will argue for the remaining one case;

(5) $\mu(b) \equiv \mathbf{G} \equiv (\mathbf{B}, \mathbf{B}_0, \mathbf{A}, \mathbf{s}, \mathbf{d}, \boldsymbol{\mu}) \in \mathbf{G}$.

Since θ_1 and θ_2 are elaborations of G, they must satisfy the Elaboration Condition (E5.1) if b is a closed box, or (E5.2) if b is an open box.

Let $\mathbf{G}' \equiv \mathbf{G}/\boldsymbol{\mu}[\theta_1(a_i)/\mathbf{b}_i \mid \mathbf{b}_i \in \text{in-box}(\mathbf{G})] = \mathbf{G}/\boldsymbol{\mu}[\theta_2(a_i)/\mathbf{b}_i \mid \mathbf{b}_i \in \text{in-box}(\mathbf{G})]$, where a_i is the i-th in-arrow of b, and $\theta_1(a_i) = \theta_2(a_i)$ by the premise of the **P**(b). Then, **G'** is bound and $\lambda(\mathbf{G}) < n$.

If **G'** is consistent, then **G'** has an unique elaboration Θ by the Induction Hypothesis, and by (E5.1) in case b is a closed box, or by (E5.2) in case b is an open box,

$\theta_1(A_j) = \cup(\Theta(\mathbf{a}) \mid \mathbf{a} \in \mathbf{d}^{-1}(B_j))$ and

$\theta_2(A_j) = \cup(\Theta(\mathbf{a}) \mid \mathbf{a} \in \mathbf{d}^{-1}(B_j))$.

Therefore, $\theta_1(A_j) = \theta_2(A_j)$ for every $A_j \in s^{-1}(b)$.

If **G'** is not consistent, b must be a closed box, and by (E5.1), $\theta_1(A_j) = \theta_2(A_j) = \perp$. Therefore **P**(b) holds. (End of Case 2)

From Case1 and Case 2, $\theta_1 = \theta_2$. (End of Induction Step)

From (I) and (II), and by the principle of mathematical induction, (P1) and consequently (P2), holds for box-graphs of arbitrary levels of nesting.

(End of Proof)

4.2 Algorithm

The following recursive algorithm τ computes the elaboration and the minimum solution of a consistent bound box-graph:

INPUT: $G \equiv (B, B_0, A, s, d, \mu)$: bound box-graph.

OUTPUT: if G is consistent then (θ, σ) else Inconsistent, where

$\theta : A \to D^+$: the elaboration of G,

$\sigma : \text{base}(G) \to D^+$: the minimum solution of G.

METHOD: Let $(b_1, b_2, \ldots, b_n)$ be a topological sorting of B based on the partial ordering defined in (B2) of §3.2.

$\tau(G) \equiv$ **begin**

for k = 1 **to** n **loop**

case $\mu(b_k)$ **of**

ϕ : $\sigma(b_k) := \cup \{\theta(a_i) \mid a_i \in d^{-1}(b_k)\}$;

$\theta(A_j) := \sigma(b_k)$ for all $A_j \in s^{-1}(b_k)$;

D^+ : **if** $\cup \{\theta(a_i) \mid a_i \in d^{-1}(b_k)\} \leqq \mu(b_k)$

then $\theta(A_j) := \mu(b_k)$ for all $A_j \in s^{-1}(b_k)$;

else return (Inconsistent); **end if**;

F^+ : **if** $b_k \in B_c \vee \theta(d^{-1}(b_k)) \in D^m$

then $\theta(s^{-1}(b_k)) := \mu(b_k)[\ \theta(d^{-1}(b_k))\ ]$;

else return (Inconsistent); **end if**;

P^+ : **if** $b_k \in B_0 \wedge$ **not** $\mu(b_k)[\ \theta(d^{-1}(b_k))\ ]$

then return (Inconsistent); **end if**;

```
G :   G' := G/μ[θ(a_i)/b_i | b_i ∈ in-box(G)];
            -- where μ(b) ≡ G ≡ ( B, B_o, R, s, d, μ) ∈ G
      if τ(G') = Inconsistent
      then if b_k ∈ B_o
            then return (Inconsistent);
            else θ(A_j) := ⊥  for all A_j ∈ s^-1(b_k); end if;
      else (σ,θ) := τ(G');
            θ(A_j) := σ(B_j)  for all A_j ∈ s^-1(b_k),
                                    B_j ∈ out-box(G');
      end if;
   end case;
end loop;
return (σ,θ);
end τ;
```

Note that, by Theorem 2, a choice of a particular total ordering as a result of topological sorting is not significant. The result of computation is independent of the choice.

5. Conclusions

Visual programming languages will be used by computer users who are not familiar with the traditional Von Neumann type machine model. It is not appropriate to expose such a low level architecture to naive end users. From that point of view, sequential programming, which is a natural consequence of Von Neumann machine model, should not be the main method of computer usage. Parallel and asynchronous computation is more natural simulation of real world activities.

We have introduced a new parallel computation model specifically designed for visual programming languages. The model does not contain the notion of variable nor state transformation. The concept of inconsistency and its range of propagation is introduced to replace the Boolean data type. It is a functional model with a dataflow flavor where no computation has side-effects. In order to make the model high level enough to abstract all implementation issues, the semantics of the model is defined declaratively rather than imperatively.

The hierarchical dataflow model is only a part of the Show & Tell Language (STL) semantics. Data query operations, recursion, iteration, and parameterizations are available in STL, but they are not included in HDM so that the model may be kept simple and that it can be compared easily with the traditional dataflow model. However, HDM is an important part of the research problem to formalize visual programming languages such as STL.

Another important problem in this research area is a formal syntax specification of two dimensional programming languages. For example, there is not yet a formal grammar for STL. An extension of phrase structure

grammar to a grammar for two dimensional languages will require an adequate definition of concatenation operation of two dimensional patterns.

Finally, there is a question of the relationship between HDM and logic programming. Consider the box-graph given in Figure 11 (b). The algorithm of §4.2 can not find the solution for this box-graph, because it is not bound and the algorithm is based on the uni-directionality of dataflow in HDM. In order to find the solution (or the elaboration) for this box-graph, we need either logical inferencing or equation solving capability. By the same token, is it possible to design 'Picture Prolog' based on HDM or its extension?

6. References

* Show and Tell is a trademark of Computer Services Corporation. Macintosh is a trademark of Apple Corporation.

[2] Raeder, G. A Survey of Current Graphical Programming Techniques. *IEEE Computer*, 18:8 (11-25), August 1985.

[3] McLain,P., and T.D. Kimura. Show and Tell User's Manual. Technical Report WUCS-86-4, Department of Computer Science, Washington University, St. Louis, March 1986.

[4] Kimura, T.D., J.W. Choi, and J.M. Mack. A Visual Language for Keyboardless Programming. Technical Report WUCS-86-6, Department of Computer Science, Washington University, St. Louis, March 1986.

[5] Davis, A.L. and R.M. Keller. Data Flow Program Graphs. *IEEE Computer*, 15:2 (26-41), February 1982.

[6] Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16:8 (57-69), August 1983.

[7] Kimura, T.D. Completion Problem and Its Solution for Context-Free Languages (Algebraic Approach). Moore School Report 72-09, University of Pennsylvania, Philadelphia, PA., May 1971.

[8] Manna, Z. *Mathematical Theory of Computation.* McGraw-Hill, New York, 1974.