Abstract Interpretation and Indeterminacy

Prakash Panangaden
Prateek Mishra
Department of Computer Science
University of Utah
Salt Lake City, UT 84112

UUCS-84-006

Abstract: We present a semantic theory that allows us to discuss the semantics of indeterminate operators in a dataflow network. The assumption is made that the language in which the indeterminate operators are written has a construct that allows for the testing of availability of data on input lines. We then show that indeterminacy arises through the use of such an operator together with the fact that communication channels produce unpredictable delays in the transmission of data. Our scheme is to use special tokens called hiatons to represent delays as measured *locally*, and then to filter out the hiatons to obtain ordinary streams. This filtering process produces indeterminate behavior at the level of ordinary streams. We indicate how this can be justified using the formalism of abstract interpretation. We show that a particular fairness anomaly does not arise.

keywords and phrases: abstract interpretation, indeterminate operators, hiatons, dataflow networks, fairness.

Table of Contents

1. Introduction

In this paper we discuss a new approach to the semantics of dataflow networks containing indeterminate operators. The major advantages of our formalism are: (i) we are able to *derive* the denotations of indeterminate operators given their implementation in terms of an imperative language resembling that of [Kahn 77], (ii) we are able to formally realise the assertion that indeterminacy arises through arbitrary delay in the arrival of data at the nodes of a network, (iii) we are able to recover the standard view of indeterminate dataflow networks through the use of abstract interpretation, (iv) we are able to reason about *fairness* properties of certain indeterminate operators and (v) we do not need to introduce an explicit *oracle* as an additional construct.

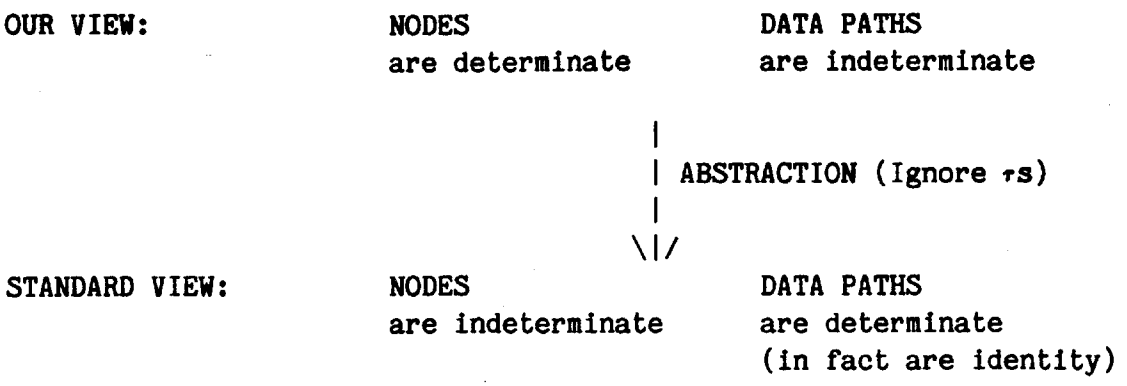
The earliest investigation into the formal semantics of such networks was carried out by Kahn [Kahn 74] who showed that if the communication between the nodes was restricted to reading from input channels and writing onto output channels then the network would be determinate. [Keller 78] proposed a new formalism in which nodes are allowed to test for the availability of data on their input arcs, thus leading to the possibility that some of the nodes are indeterminate when viewed as acting on streams of data tokens. We give a formal semantics to this extended Kahn-Keller language.

Our formalism incorporates two recent developments. The first is the use of a special token called a *hiaton* [Park 82] to represent a delay in the propagation of data. We shall use the symbol τ to represent a hiaton. Thus, a typical stream will contain data values interspersed with τs. The interpretation of such a stream will be clarified in the next paragraph. The virtue of introducing such a special token is that we can give definitions of indeterminate operators on streams as determinate functions on τ-enriched streams. The second recent development that we make use of is the idea of an "abstract" interpretation [Mycroft 81] [Mishra 84]. The key idea here is to map the semantic domain

onto a (in some suitable sense) simplified domain and to reinterpret the language constructs in terms of the simplified domain. The technical details of abstract interpretation will be discussed in the next section. We will use abstract interpretations in a slightly different way than it has traditionally been used. In our case the original domain will be the domain of τ-enriched streams (τ-streams) and the simplified domain will be the standard domain of streams. Via the abstraction process, the determinate functions over the domain of τ-streams are reinterpreted as indeterminate operators over the domain of standard streams.

In interpreting the τ symbols we make the following assumptions. Each node has an internal clock which is not synchronized in any way with the internal clock of any other node. The ticks of the clocks mark time periods during which the basic communication actions occur. These basic actions are: read from an input channel, write onto an output channel and test a channel for availability of data. Thus a τ on the input stream would represent the non-appearance of data on that channel during a cycle. The τ symbols thus convey information about the relative rates of appearance of data at a particular node. The nodes that we model are perfectly determinate with respect to τ-enriched streams, as no additional indeterminate primitives, such as internal choice are allowed. The meaning of a particular node is defined by a function from τ-enriched streams to streams, as the only explicit indeterminate behavior that we incorporate at this level is that the channels between nodes insert τ symbols into the output stream in an unpredictable way. Thus, the τ symbols cannot be used to obtain effective synchronization between different nodes. One novelty of our approach is that we have succeeded in incorporating timing considerations into our theory without having to introduce any *global* clock.

The formalism we present may be visualized through the following diagram:



The rest of the paper is organized in the following way. In the next section we discuss the basic formalism of abstract interpretation. In section 3 we introduce a simple programming language for writing implementations of the nodes and we give its semantics in terms of τ-enriched streams. In the following section we abstract this interpretation to the domain of ordinary streams and show how indeterminacy arises. The remaining sections are devoted to certain applications particularly to the analysis of fairness. In this context we are able to show that a particular fairness anomaly does not arise.

2. Abstract Interpretation

Abstract interpretation provides a general framework for the static inference of properties of programs. Typically such inference is of use in program optimization, transformation and (weak) verification (i.e. where termination issues are not addressed). Abstract interpretation was first developed by Cousot & Cousot [Cousot 77, Cousot 81] to aid in the analysis of imperative programs. In this setting programs are modelled as flowcharts; static inference is expressed as an abstraction of the "collecting interpretation" - the natural lifting of the standard semantics from values to *sets* of values.

In developing the abstract interpretation of applicative programs, Mycroft [Mycroft 81, Mycroft 83] observed that the framework of Cousot & Cousot was useful only for inference schemes which were weakly correct; termination cannot be expressed in their framework. In place of the powerset based formulation as used by the Cousots, a powerdomain based formulation is necessary. This view was further developed in [Mishra 84] where the abstract interpretation of non-flat "stream" domains was considered.

Our use of abstract interpretation is rather different. We use the formalism to show that standard stream based programming with indeterminate operators can be viewed as an **approximation** to programming with τ-streams, which incorporate a certain degree of timing information. This allows us to pinpoint the phenomenon of indeterminacy as arising from ignoring the explicit timing information and from the fact that the data lines introduce randomness into the relative speeds of transmission of data. Furthermore, our formalism does not introduce oracles as an additional construct, as the abstraction map serves the role of an oracle. Among the advantages of our approach is that it articulates in a precise fashion the relation between indeterminacy and imprecise information about the computational details. This has frequently been expressed in the literature in an intuitive fashion but has never before captured precisely. Another benefit is that the introduction of explicit delay allows us to reason about **fairness**, something which is very difficult otherwise.

The domains of interest in the present work are the domain of τ-streams, written TS, and the domain of streams. These can be defined as:

```
TS = stream[integers + τ]

S = stream[integers]
The connection between the two is established by:

Ab: TS --> S
where
Ab: nil = nil
Ab: τ^x = Abs:x
(Here, and in what follows, ^ represents the infix stream constructor,
"followedby" or "cons".)
```

The domain of ordinary streams over a set of data values V, is the solution of the recursive domain equation

$$S = []+V+(V \times S)$$

and has the usual prefix ordering. The domain of τ-streams is the solution of the recursive domain equation

$$TS = []+(V \cup \{\tau\})+((V \cup \{\tau\}) \times TS)$$

once again with the prefix ordering. It is easy to see that the function **Ab** defined above is a continuous map from TS to S.

We cannot use the published accounts of abstract interpretation for our analysis, since the best results available apply only to domains of finite height [Mycroft 83]. We use instead the formalism of [Panangaden 84], which is applicable for general domains. This formalism uses categories, rather than complete partial orders, to model computability concepts. The idea is that the existence of a morphism from an object X to another object Y expresses the fact that X might "improve" to Y. However, approximation is no longer antisymmetric, as there might be morphisms in both directions between two objects. This broadening of the concept of approximation is necessary when dealing with sets of values. The fact that we no longer have a partial order may appear to deprive us of the concept of limits, based as it is on the notion of a least upper bound; in categories

we can, however, make use of the theory of direct limits [Arbib 75] to serve this purpose. Using morphisms to express approximation between objects allows us to distinguish *different ways in which two objects may approximate each other*, a distinction crucial to the formalism. Further details may be found in [Panangaden 84].

Given domains D and E and continuous functions from D to E, we wish to extend these functions to "sets of values" in such a way that the extension process (lifting) is continuous and the lifted functions are also continuous. The first step is to redefine the original functions so that they now run from D X ? to D X E, where ? is the one point domain. Given a function f:D --> E we define f^g:D X ? --> D X E by $f^g(<d,?>)=<d,f(d)>$. Clearly f^g is continuous and monotonic iff f is. We now define the category **PG**(D,?); the objects are all subsets of D X ?, the morphisms are *increasing* maps. Similarly, we define the category **PG**(D,E). A function f^g from D X ? to D X E is now extended to a *functor* F from **PG**(D,?) to **PG**(D,E) in the following way:

1. The action of F on an object X is
$F(X) = \{<d,f(d)> | <d,?> \epsilon X\}$
2. The action of F on a morphism κ from X to Y is to yield a morphism
$F(\kappa)$ from F(X) to F(Y) defined by:
if $\kappa(<d,?>)=<d',?>$ then $F(\kappa)(<d,f(d)>)=<d',f(d')>$.

The domains D X ? and D X E can be embedded in the obvious way into **PG**(D,?) and **PG**(D,E) respectively. In [Panangaden 84] it is shown that these embedding are continuous.

An abstract interpretation can now be defined in the following way. First "simplified" domains A and B are introduced, then **PG**(A,?) **PG**(A,B) are constructed in the same way as before. Now the simplification is expressed via abstraction maps γ_1 and γ_2 from **PG**(D,?) to **PG**(A,?) and from **PG**(D,E) to **PG**(A,B) respectively. The requirements for an abstraction map to be acceptable are that if a morphism exists from X to Y then it should be possible to find a morphism from $\gamma_{1,2}(X)$ to $\gamma_{1,2}(Y)$. This is a slightly weaker condition

than requiring them to be functors. The abstraction maps should also satisfy a "weak" continuity requirement which is explained in [Panangaden 84].

In our application we will use TS and S as the domains D and E respectively and S for both A and B. Thus our abstraction is to ignore the timing information contained in τ-streams. In section 4 we briefly discuss the formalism sketched in this section as it applies to streams and τ-streams.

3. The Language

We present a simple imperative language for writing implementations of the nodes (operators) in a dataflow network. The language is essentially the same as the language presented in [Kahn 74] with the addition of a construct called "poll" [Keller 78] which tests for the availability of data on the input channels. The communication between nodes is via input and output channels. These channels are viewed as unbounded queues. A program for a particular node contains a declaration of the input and output channels. The interaction between a node and its channels is carried out by read, poll and write primitives. A read operation on a channel removes a token from that channel and puts the token in the local store. Similarly, a write operation takes a token from the local store and copies it onto the output channel. The poll operation on an input channel is boolean valued and returns false when there is no token available, in other words, when the first item in the input stream is a hiaton. We assume that there is always at least a single hiaton on every input channel to start with. The remaining language primitives allow one to manipulate the local store and construct arithmetic and logical expressions in the standard fashion.

The syntax of the language is given by the following grammar:

<program>::=<declarations><body>
<declarations>::=**input channel**:<identifier>|**output channel**:<identifier>

```
<body>::=<variable declarations><statements>
<statements>::=Λ|<statement>;<statements>
<statement>::=if<boolean>then<statements>else<statements>|
              while<boolean>do<statements>|
              <identifier>:=<expression>|
              read from<identifier> into <identifier>|
              write<identifier> onto <identifier>
<boolean>::=poll <identifier>|...
```

The syntax for expressions and booleans is assumed to be standard.

An example program in this language is shown below [Keller 78].

```
input channel:a,b
output channel:c
variable:x
while true do
if poll a then
    read from a into x;
    write x onto c
    if poll b then
        read from b into x;
        write x onto c
    else
        continue;
else
    if poll b then
        read from b into x;
        write x onto c
    else
        continue.
```

This program [Keller 78] implements a fair merge operation. It is clearly a determinate program when viewed as acting on τ-enriched streams, but when viewed as acting on ordinary streams it becomes indeterminate. This particular style of programming language has been implemented by [Tanaka 83] on the REDIFLOW simulator [Keller 83].

We shall define a meaning function μ for this language assuming that μ is already defined for the purely sequential part of the language. We then give semantics for read, write and poll. The meaning function will map statements to functions from

environments to environments. The environment is a function that maps the local variables to values and the input channels to τ-streams and the output channels to ordinary streams. The expressions get mapped to functions from environments to values. The propagation of values along the data paths is modelled by the extraction of the appropriate output stream from the environment and the insertion of an unpredictable number of delay tokens via a closure operation. Thus the input to the next node will be a set of possible input streams rather than a single stream. We define the action of a node on a set of possible input streams as the pointwise extension of its action on individual streams. We are suppressing the full mathematical details in this discussion as they would clutter up the formalism. However, in [Panangaden 84] we shall give the mathematical details involving the categorical collecting semantics.

The semantics of the poll construct is as follows:

μ(**poll** a)(env)=<newenv,val>
where
val = false if first(env(a))=τ , val=true otherwise
and if val = false then newenv=env(a|-rest(env(a))) else newenv=env.
The modifications to the environment are indicated by the |- symbol, thus
env(a|-rest(env(a))) means that the binding of a in the environment is
changed to be bound to rest of the old binding.

Thus the poll construct essentially detects the presence of hiatons. If there is a hiaton on the channel being polled then the poll operation consumes it, this ensures that poll is in fact time sensitive. The **read** construct is hiaton insensitive, thus it consumes all the leading hiatons from a channel unless it is guarded by a poll operation. The **write** construct will not produce any hiatons at all. The semantics of these constructs are as follows:

μ(**read** a into x)(env) =
env(a|-rest(strip_τ(env(a)));x|-first(strip_τ(env(a))))

μ(**write** x onto b)(env) = env(b|-addtoend(env(b),env(x))),

where the addtoend function is defined as:

addtoend(x,y)=rev(y ^ (rev(x))).

It is now easy to see that the action of the merge node is given by the following function:

μ(merge)(env)=if first(env(a))=τ then f(newenv1) else
f(newenv2)

newenv1 = env(a|-rest(env(a)))
newenv2 = env(a|-rest(env(a)),c|-addtoend(env(c),first(env(a))))

f(env) = if first(env(b))=τ then μ(merge)(env(b|-rest(b))) else
μ(merge)(env(b|-rest(env(b)),c|-addtoend(env(c),first(env(b))))).

The next issue that we discuss is how to derive the meaning of a network of operators given the meaning of the individual nodes. To do so it is necessary to discuss the action of a data line in transmitting a stream from one node to another. In the case of ordinary streams a data line is merely the identity function. On τ-streams, however, a data line causes an unpredictable delay in the transmission of data values and hence causes the insertion of arbitrarily many hiatons. The action of a data line is thus a function from ordinary streams to **sets of** τ-streams. This action is most easily expressed in terms of the inverse of the abstraction map. Recall, from section 2, that the abstraction map is a map from τ-streams to streams which removes all the hiatons from the τ-stream. The inverse map, therefore, maps a stream to the set of τ-streams that result by arbitrary insertion of hiatons into the stream. Thus the action of a data line on a stream s is simply $\mathbf{Ab}^{-1}$ (s). The action of an operator **O** with input line **a** and output line **b** is thus

$\mathbf{Ab}^{-1}(\mu(\mathbf{O})\ (t))$,
where t is the input τ-stream on **a**.

If the output line **b** is connected to the input line of another operator then its meaning function acts on each member of the set of τ-streams produced by **O**. Thus composition of the meaning functions of operators is effected via the action of $\mathbf{Ab}^{-1}$. This extends in the obvious way to operators that have several input and output lines.

4. Abstraction in Our Context

In our approach we have two structures to reason about, the domain of timed streams and the domain of ordinary streams. As has been discussed in sec. 2, the formal justification of our abstraction is done via the powergraph construction. In other words the domain of τ-streams must be first viewed as a category. We must then show how to abstract to the category of streams.

The category of τ-streams is defined in the following way:

Def 4.1:The category **PG**(TS,?) has as objects *all* subsets of TS X ?, where ? is the one point domain and TS is the domain of τ-streams. The morphisms are maps between objects that satisfy: if κ is a morphism from X to Y and if $\kappa(x)=y$ then $x <= y$ in the partial order of the domain D X ?.

Maps from TS to S are modelled by introducing the new category **PG**(TS,S):

Def 4.2:The category **PG**(TS,S) is defined as: (1) the objects are all subsets of TS X S, (2) the morphisms are *increasing* maps between objects.

First of all maps from TS to S are reinterpreted as maps from TS X ? to TS X S as described in section 2. They are then reinterpreted as *functors* from **PG**(TS,?) to **PG**(TS,S). In [Panangaden 84] it is shown that this process is mathematically well defined, in other words continuity of the lifted functions is assured and the lifting process is itself continuous.

The abstraction is described as a map from the objects of **PG**(TS,S) to the objects of **PG**(S,S) and another map from the objects of **PG**(TS,?) to the objects of **PG**(S,?). The maps in our case are obtained by extending the map **Ab** to sets in the natural pointwise fashion. It is clear that channels define indeterminate functions from S to TS and that these become identity functions under abstraction. More precisely the abstraction map, γ_2, from **PG**(TS,S) to **PG**(S,S) is given by:

$\gamma_2(X) = \{<s_1,s_2>|<ts_1,s_2>\epsilon X$ and $\mathbf{Ab}(ts_1(ts_1)=s_1$.

The abstraction map from **PG**(TS,?) to **PG**(S,?) is defined analogously. It is easy to check

that these abstraction maps satisfy the conditions required of acceptable abstractions as defined in section 2.

5. Fairness

In this section we present some preliminary remarks about the application of our formalism to reasoning about fairness. One possible application of our formalism is reasoning about fairness. The original motivation for the introduction of hiatons in [Park 82] was to reason about fairness. We have not articulated the distinctions between various kinds of fairness in our formalism but we have some preliminary results.

One question that naturally arises, now that we can derive the denotations of indeterminate operators, is whether a given node is fair. An obvious answer is the following. Suppose that an operator loops repeatedly over a piece of code in the Kahn-Keller language and suppose that in each loop it tests *every* input channel for available data, then, when abstracted, the resulting indeterminate operator is fair. Furthermore if the loop is endless, the operator will be infinity-fair in the sense of [Park 82]. It is now immediately clear that the merge program of section 3 is fair.

This result is of course only indicative of the flavor of possible results in this direction. We are currently working on developing non-trivial *static* tests for fairness properties.

In [Keller 78] the merge anomaly was first introduced. We have recently observed that the same example illustrates a fairness anomaly. Consider a merge operator with its output arc connected back to one of its input arcs. Suppose the stream "ab" is fed in at the remaining input arc. Then, assuming that the operator is fair, the set of possible output streams are $\{a^n b(ab)^{\infty} | n>0)\}$. If we take the "limit" we obtain the possible stream a^{∞}; this corresponds to the situation where the first a cycled through infinitely often before the b token appeared. Thus the resulting network is not fair. We need to express

a *finite delay condition* that ensures that this is not possible [Karp 69]. In our framework, we can express the finite delay property required in an obvious fashion, we ensure that there is always at least one hiaton before the a reappears as an input token, then eventually the merge must get the b token. In terms of τ-streams the fairness anomaly cannot occur, and when we abstract the fixed point set resulting from the iteration through the merge operator, we will not have the pathological a^{ω} stream.

6. Conclusions

To summarize, we have presented a formalism that allows one to derive the behavior of indeterminate operators when they are implemented in a lower level language. Typically, analyses of indeterminacy focus on deriving behaviors of networks from the behaviors of the nodes. We have used a new approach to abstract interpretation that allows us to put our theory on a semantically sound footing and we have illustrated how our theory could be useful for reasoning about fairness.

Several other applications are also possible. Two directions we are pursuing are to develop static tests for determinacy and for fairness and to study deadlock properties of networks containing indeterminate operators [Wadge 79].

An interesting observation is that while typically, abstract interpretation is used for justifying static analysis by simplifying the domain, we have used it in the reverse fashion; that is, we have enriched the standard domain to obtain a useful and interesting model.

7. Acknowledgements

We would like to thank Robert Keller, Uday Reddy and Esther Shilcrat for helpful discussions. This research was supported by a grant from IBM Corporation, by CER grant # MCS-8121750 awarded to the Dept. Of Computer Science at the University of Utah and by a University of Utah Graduate Fellowship awarded to Prateek Mishra.

References

[Arbib 75] Arbib M. A., Manes E. G.
Arrows, Structures and Functors.
Academic press, 1975.

[Cousot 77] P. Cousot and R. Cousot.
Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.
POPL IV :238-252, Jan, 1977.

[Cousot 81] P. Cousot.
Semantic Foundations of program analysis.
Prentice-Hall, 1981, pages 303-342.

[Kahn 74] Kahn G.
The Semantics of a Simple Language for Parallel Programming.
In *Proc. IFIP 1974*, pages 471-475. 1974.

[Kahn 77] Kahn G., McQueen D.
Coroutines and Networks of Parallel Processes.
In B. Gilchrist (editor), *Information Processing 77*, pages 994-998. 1977.

[Karp 69] Karp R. M., Miller R.
Parallel program schemata.
JCSS , May, 1969.

[Keller 78] Keller R.M.
Denotational Models for Parallel Programs With Indeterminate Operators.
In E.J.Neuhold (editor), *Formal Descriptions of Programming Concepts*, pages 337-365. North-Holland, Amsterdam, 1978.

[Keller 83] Keller R. M.
Users' Manual for Function Equation Language.
AMPS Technical Memorandum 7, U of Utah, July, 1983.

[Mishra 84] P. Mishra, R. M. Keller.
Static inference of properties of applicative programs.
In *POPL XI, Salt Lake City.* January, 1984.

[Mycroft 81] A. Mycroft.
Abstract Interpretation and Optimising Transformations for Applicative Programs.
PhD thesis, University of Edinburgh, December, 1981.

[Mycroft 83] Mycroft A. and Nielsen F.
Strong Abstract Interpretation Using Powerdomains.
In Diaz J. (editor), *Automata, Languages and Programming*, pages 536-547. EATCS, July, 1983.

[Panangaden 84] Panangaden P., Mishra P.
A Category Theoretic formalism for Abstract Interpretation.
Technical Report UUCS-84-005, University of Utah, May, 1984.

[Park 82] Park D.
The Fairness Problem and Nondeterministic Computing Networks.
In *Proc. 4th Advanced Course on Theoretical Computer Science.*
Mathematisch Centrum, 1982.

[Tanaka 83] Tanaka J. and Keller R. M.
S-code Extension in FEL.
AMPS Technical Memorandum 10, U of Utah, July, 1983.

[Wadge 79] W. Wadge.
An Extensional treatment of Dataflow Deadlock.
Springer-Verlag, 1979, pages 285-299.