A Generative Programming Approach to Interactive Information Retrieval: Insights and Experiences

Saverio Perugini[1] and Naren Ramakrishnan[2]

[1] Department of Computer Science
University of Dayton, OH 45469-2160 USA
saverio@udayton.edu
http://homepages.udayton.edu/~perugisa

[2] Department of Computer Science
Virginia Tech, VA 24061-0106 USA
naren@cs.vt.edu
http://people.cs.vt.edu/~ramakris

Project website: http://oot.cps.udayton.edu

Abstract. We describe the application of generative programming to a problem in interactive information retrieval. The particular interactive information retrieval problem we study is the support for 'out of turn interaction' with a website – how a user can communicate input to a website when the site is not soliciting such information on the current page, but will do so on a subsequent page. Our solution approach makes generous use of program transformations (partial evaluation, currying, and slicing) to delay the site's current solicitation for input until after the user's out-of-turn input is processed. We illustrate how studying out-of-turn interaction through a generative lens leads to several valuable insights: (i) the concept of a web dialog, (ii) an improved understanding of web taxonomies, and (iii) new web interaction techniques and interfaces. These notions allow us to cast the design of interactive (and responsive) websites in terms of the underlying dialog structure and, further, suggest a simple implementation strategy with a clean separation of concerns. We also highlight new research directions opened up by the generative programming approach to interactive information retrieval such as the idea of web interaction axioms.

1 Introduction

Generative programming has been typically been applied to problems at the crossroads of programming languages and software engineering such as modularizing cross-cutting concerns, synthesizing programs from formal specifications, and automatically generating program documentation. We describe here a novel application of generative programming to a problem in interactive information retrieval [1].

1.1 Motivating Example

Everybody has experienced the frustration in interacting with automated information systems where the system does not let the user progress through the dialog without answering a currently posed question. For instance,

1 **System:** Welcome to the automated flight reservation system.
2 **System:** Please say the date on which you wish to travel.
3 **Sallie:** I'd like to fly from New York to Brussels next week.
4 **System:** Sorry, I didn't understand. Please specify a date.
5 **Sallie:** If you can tell me available dates, I can choose.
6 **System:** Please say the date on which you wish to travel.
7 **Sallie:** [Hangs up]

The mental mismatch between Sallie's conception of the task and the system's design is manifest in the above interaction. The system is expecting a date in Line 3 whereas Sallie specifies her choice of source and destination cities. Even though this information is going to be relevant further into the interaction, the system insists on specifying date before going further. Similar inconveniences happen while interacting with websites. A site presents hardwired choices of hyperlinks to pursue and even though the user's input is pertinent and probably solicited deeper in the site, there is no way for the user to circumvent the given navigation structure.

Our solution to the above situations, where the user cannot answer a currently posed question, but does have some other information pertinent to the task at hand, is to provide a capability for *out-of-turn* interaction. For instance, we would provide a capability for the user to speak something into the browser, and in this way supply out-of-turn input. Such 'unsolicited reporting' has been recognized [2] as a simple form of *mixed-initiative interaction*, a dialog management strategy where the two participants take turns exchanging the initiative. Using out-of-turn interaction, the user is empowered to complete an information-finding task in the manner that best suits her conception. Moreover, we show that out-of-turn interaction, irrespective of when it happens, can be supported uniformly by a generative programming approach. A website that currently provides a hardwired choice of completion options can be automatically converted into one that supports out-of-turn interaction!

The idea behind our approach is quite simple: we liken out-of-turn interaction to non-sequential evaluation of a computer program, e.g., partial evaluation. We model an information seeking interaction as a computer program so that user inputs correspond to values for program variables (ref. Fig. 1, left). When the user provides input in the order in which they are requested, we are sequentially evaluating the program, i.e., interpreting it. In a web hierarchy, this would correspond to plain browsing. When the user provides out-of-turn input, we jump ahead to nested program segments that involve that input and simplify them out via partial evaluation. By employing sequences of such interpretations and partial evaluations, we can support complex interactions that involve both responsive as well as out-of-turn inputs.

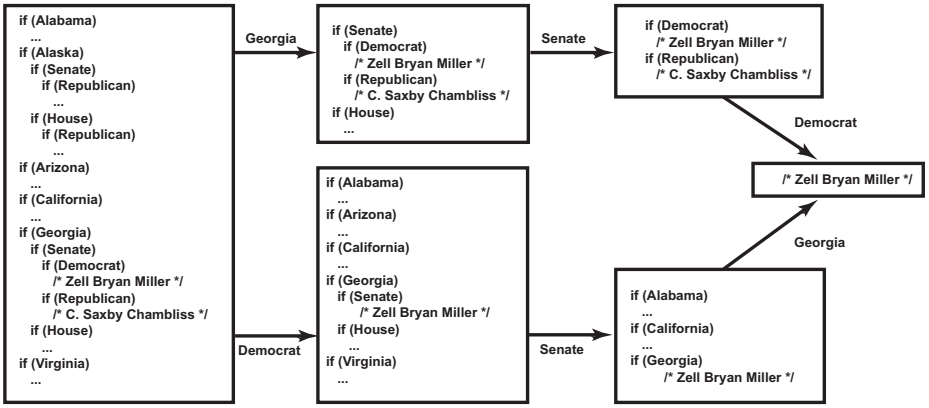

Fig. 1. Staging web interactions using program transformations. The top series of transformations mimic an in-turn (i.e., browsing) interaction sequence with the user specifying (Georgia: Senate: Democrat), in that order (ref. Fig. 2, left). The bottom series of transformations correspond to an out-of-turn interaction sequence where the user specifies (Democrat: Senator: Georgia), in that order (ref. Fig. 2, right). Notice that we can stage both interaction sequences here with the *same* program transformation! *All* programs shown here are *partial evaluations* of the starting program (left).

1.2 Implementing Out-of-turn Interaction Generatively

Now, since a given program can be transformed in numerous ways, the designer need only write the program in one way but the use of program transformations enables us to realize all possible interaction sequences. Further, since partial evaluation subsumes interpretation, there is no need to distinguish between an in-turn or out-of-turn input. This enables a simple implementation strategy with a clean separation of concerns. An input, supplied using any of a variety of user interfaces, is communicated to a server where it is used to partially evaluate a program. The resulting program is rendered as a website and presented back to the user. Fig. 1 depicts these ideas using Project Vote Smart (PVS; `www.vote-smart.org`), a website which indexes the webpages of the US Congressional Officials and asks a user to make a selection for state, branch of Congress, and party, *in that order*, to access an official's page. Fig. 2, left and right, illustrates how the sequences staged by the top and bottom series of program transformations in Fig. 1, respectively, might be rendered on the web.

Our generative approach makes enabling out-of-turn interaction in an existing website a fairly mechanical process. The approach requires four components: a representation, transformer, out-of-turn interaction interface, and generator. A representation of the site's hyperlink structure, such as that in Fig. 1 (left), can be extracted from the original site and stored in its server from which it will be transformed to stage user interaction. Such a representation can be easily generated from a depth-first traversal of the site using either an off-the-shelf web crawler or web scripting languages (e.g., Python) to build a customized bot. The

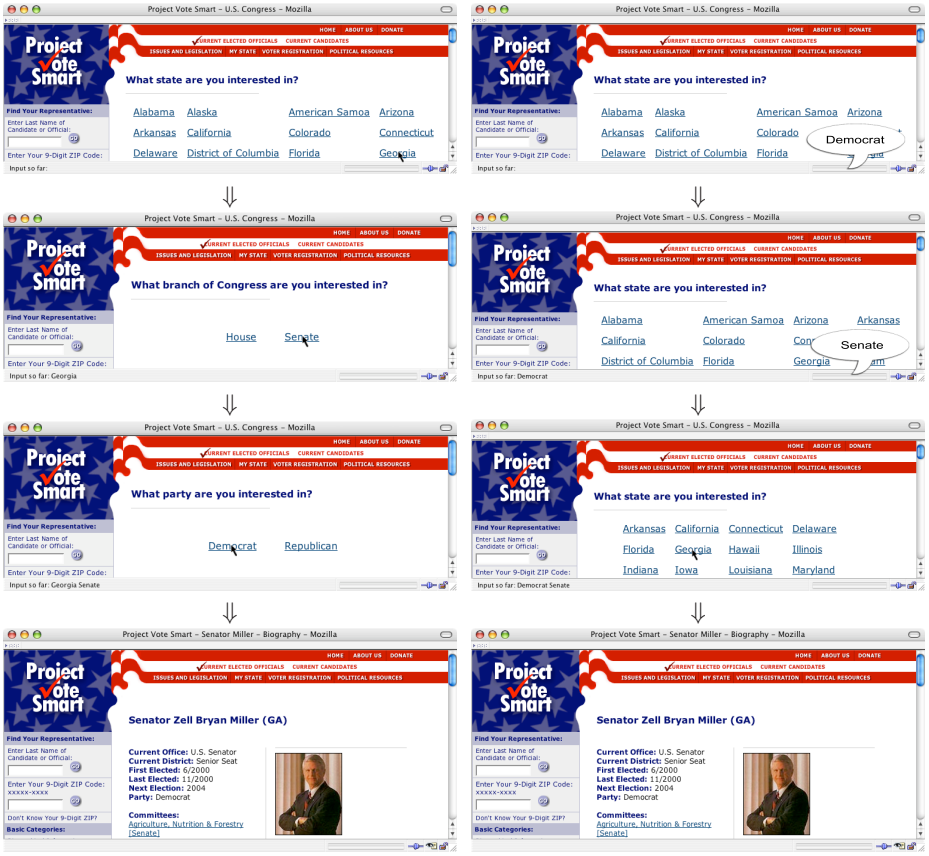

Fig. 2. Retrieving the webpage of the Senator Miller in PVS. (left) In-turn interaction sequence: the user specifies values for relevant politician attributes by progressively clicking on the presented hyperlinks (Georgia: Senate: Democrat), in that order. (right) Out-of-turn interaction sequence: user specifies (Democrat: Senate: Georgia), in that order, using out-of-turn interaction via voice.

out-of-turn interaction interface captures and communicates the user's out-of-turn input (i.e., a string) to a web server. We have built an out-of-turn interaction toolbar interface, called *Extempore*, using XUL (XML User Interface Language). *Extempore* is embedded into a traditional web browser as a plug-in [3]. We also have implemented a voice interface using SALT (Speech Application Language Tags) to capture out-of-turn speech utterances (illustrated in Fig. 2, right). A server-side program or web service transforms the representation given a set of (in-turn or out-of-turn) user input terms (communicated via a hyperlink click or the out-of-turn interface). Lastly, the generator produces a webpage containing hyperlink labels corresponding to the variables at the topmost level of nesting in the representation.

Initially we simply run the representation through the generator to create the top-level page of the site. The resulting webpage is aesthetically identical to the original site's homepage except that each hyperlink now represents a request to invoke the transformation operator on the representation wrt the hyperlink's label rather than a request for a static page. Once this initialization is complete, a communicate-transform-generate loop responds to each user interaction (hyperlink click or out-of-turn input). The user *communicate*s an input using the available interaction interfaces (hyperlinks or the out-of-turn interface). This input is used to *transform* the representation. Then the *generator* produces the resulting page from the new representation. Notice that there is no longer a need to store and retrieve any static pages. The current page is always generated dynamically from the the topmost level of nesting of the mostly recently transformed representation. The malleability of the representation stages the interaction and provides the illusion of a website containing a hardwired hierarchy of hyperlinks. When the representation reduces to the modeling of a single `page`, the user is redirected to that webpage. Note also that the representation is the only site-specific component in our framework.

1.3 Outline

Our research began by exploring the use of partial evaluation to transform representations of websites, for realizing out-of-turn interaction [3]. One of the first lessons we learned was that program transformers have a novel use (hitherto unexplored) as *stagers*, i.e., devices to mediate and manage interaction between two entities. In this sense, a partial evaluator is not just a pre-processor before a compiler, it is an active participant in an interaction loop between the human and the information system. This led us to investigate other program transformers (e.g., currying) and study their staging properties. We are now able to develop complex (web) dialogs as compositions of these primitive stagers [4], especially those involving mixed-initiative interaction. This paper begins by presenting these notions. Studying dialog simplification in this context then leads us to an improved understanding of web taxonomies. Next, we present new web interaction techniques and associated interfaces that exploit properties of taxonomies and which allow us to support complex dialogs. We conclude by introducing the idea of web interaction axioms and their potential role in interactive systems.

2 Related Research

Concepts from generative programming (partial evaluation [5], currying [6], program slicing [7], and continuations [8]), have been traditionally studied and employed in systems such as compilers and debuggers. While there are established and effective models for classical information retrieval (e.g., vector-space [9]), models for solutions to interactive information retrieval (IR) problems are in their infancy. Generative programming suggests helpful metaphors for developing such models. However, generative programming is under-explored in the interactive IR community.

Belkin *et al.* introduced the idea of an 'interaction script' [10] which can be thought of as a program for interaction, though expressed in English rather than program codes and is only intended to be sequentially evaluated. Slicing [11], and source-to-source rewrite rules [12] have been used to restructure web applications. Graunke *et al.* [13] describe an approach to automatically restructure batch programs for interactive use on the World Wide Web. An important issue addressed is maintaining state across web interactions which use the stateless HTTP protocol. Their approach involves first-class continuations from programming languages [8], e.g., via the `call/cc` (call-with-current-continuation) facility provided by Scheme. Since first-class continuations can be saved and resumed, they are an ideal construct for saving and restoring state between user interactions over the web. Using a similar idea based on continuations, Queinnec [14] developed a model for a web server intended to address state maintenance problems caused by connections terminated prematurely, pressing the 'back button,' and window cloning. Lastly, Quan *et al.* [15] explore the idea of using continuations and currying to postpone, save, and resume interactions with intrusive dialog boxes, including partially-filled ones, in traditional application software, such word processors and e-mail clients. The common theme of these efforts, including our research, is the appeal to concepts from programming languages to achieve a rich and expressive form of a human-computer interaction. Our work differs from all these efforts in its focus on out-of-turn interactions (and dialogs involving them). We believe that the generative programming approach presented here suggests useful metaphors for developing interactive information systems and also lends insights into representations for complex dialogs.

3 Web Dialogs

In studying the nature of dialogs supported in our framework, we started to think of a program transformation in terms of the number of interaction sequences it is capable of staging, which we refer to as the transformer's *interaction paradigm*. For example, our use of partial evaluation in PVS is capable of staging interaction sequences representing all permutations of state, branch, party or, in other words, 3,240 ($= 540 \times 3!$) sequences. PVS has 540 paths from its root to each leaf corresponding to the 540 members of the US Congress. In general, a partial evaluator can support $m \times n!$ sequences assuming that each of the m paths through the site has a consistent dialog length of n. Studying program transformers via the number of sequences they support revealed that partial evaluation could stage $m \times n!$ sequences in a given site, but no less. In other words, while partial evaluation can support all orders of supplying inputs, it cannot *enforce* an order. This property prevented us from, e.g., staging dialogs involving state, branch, party, where the party information *must* be supplied second. This 'all or nothing' nature of partial evaluation arises because, without factoring a program into multiple units, there is no way to prevent expressions containing particular remaining variables from being simplified by a partial evaluator.

This compelled us to develop a dialog notation, where the specific inputs (e.g., Alabama, Senate, Democrat) are abstracted into their categories (e.g., state, branch, party) and used as dialog slots in the context of a program transformer. The syntax of our notation uses the abbreviation of a program transformation (e.g., PE for partial evaluator) over a sequence of such slots. For example, we represent a dialog where values for state, branch, party may be communicated in any order as $\frac{PE}{\text{state branch party}}$. Such an expression succinctly compacts a set of interaction sequences; contrast this with the programs in Fig. 1 where the inputs are woven into the dialog structure. Next we incorporated additional program transformers in order to achieve a finer level of control over the number of interaction sequences stagable. For example, a *currier* stages a different number of sequences than a partial evaluator, i.e., $\frac{C}{\text{state branch party}} \neq \frac{PE}{\text{state branch party}}$. The former only permits the user to supply a *prefix* of the remaining dialog options at any point in the interaction, whereas the latter makes no such restriction. Next we can begin to nest program transformers on top of each other to create complex dialogs (i.e., dialogs composed of smaller dialogs, or subdialogs). For instance, $\frac{PE}{\frac{PE}{a\ b}\ \frac{PE}{c\ d}}$ precludes sequences such as $\prec c\ a\ b\ d \succ$. This notation provided a concise way to specify complex dialogs (i.e., much more compact than enumerating each individual interaction sequence to be supported). In addition, using a small set of reduction rules [4], which indicate how any dialog (described in this notation) should be simplified each time a user supplies an input, we are able to stage a variety of web dialogs in our generative framework.

4 An Improved Understanding of Web Taxonomies

Dialog simplification in the staging transformations framework can be viewed as pruning branches of a website based on user input. This led us to investigate functional dependencies in information hierarchies, a concept which implicitly captures what should remain and what should be pruned out when a user supplies input.

4.1 Functional Dependencies on the Web

Intuitively, an FD of the form $x \rightarrow y$ exists in a website when *all* paths (from the root to a leaf) through the site containing x also contain y. Notice that $x \rightarrow y$ does not necessarily mean that $y \rightarrow x$. In the generative approach, since representations change dynamically after every interaction, the set of FDs satisfied by a site also changes dynamically. As some paths through the site are pruned out by partial evaluation, new dependencies emerge. For instance, communicating 'Senate' to PVS out-of-turn at the top level, causes the 'Virginia $\rightarrow$ Republican' FD to emerge. This FD is not present in the original site because not all politicians in Virginia are Republicans (but the Senators are). In the untransformed PVS site, there are 129 FDs!

The set of FDs a site satisfies can be mined from a relational representation of the paths through the site, where each tuple in the relation corresponds to a

path, using standard algorithms from association rule mining [16]. We added a mining component to our generative framework to discover FDs. Since FDs must be re-computed at every step, we would like to optimize the process of mining them. While non-intuitive, it happens to be helpful to postpone computing the *current* set of FDs until *after* the user's input has been processed. In other words, rather than discovering that the 'Virginia → Republican' FD (among many others) exists after the user supplies 'Senate', only compute that FD if and when the knowledge of its existence is required (e.g., if the users supplies 'Virginia' next!). This lazy discovery not only prevents us from computing FDs that are irrelevant to the task at hand, but also results in a more efficient and simple mining procedure. For example, if the user does supply 'Virginia' next, we need only observe that the only party option remaining is 'Republican' to conclude that the 'Virginia → Republican' FD held *before* the input 'Virginia' was processed – a procedure more efficient than examining all pairs of terms (which co-occur) *a priori* as candidates for potential FDs.

FDs serve multiple uses in the generative framework. First, and most obviously, they suggest a simple strategy to perform *input expansion*. For instance, if a user communicates 'Washington, DC' at the top level of PVS we can safely expand this input to 'Washington, DC House Democrat', without changing the semantics of the request, because the 'Washington, DC → {House, Democrat}' dependency exists. Second, the exploitation of FDs results in cleaner representations, by consolidating series' of nested conditionals without else clauses, thereby relieving the user from having to click through several pages, each with only one link.

Further, we can generalize the notion of FDs to say that an FD of the form $x \rightarrow y$ exists in a website when at least $t\%$ of the paths through the site containing x also contain y. Using FDs of this sort for input expansion makes interactive IR *approximate*. Approximate interactive IR is important as it enables a host of new and compelling queries and suggests novel user interfaces. We shall have more to say about these two items in section 5 when we discuss new interaction techniques. In summary, we have illustrated how our generative approach led to the concept of a web FD and a new way to conduct query expansion on the web which together led to approximate interactive IR on the web.

4.2 Levelwise and Non-levelwise Taxonomies

Thus far, we have focused on out-of-turn inputs in a levelwise taxonomy, where each input (e.g., Virginia) addresses a distinct information category (e.g., state). With very minor modifications, we can extend our approach to work with non-levelwise taxonomies – those where no such organization exists. For instance, consider the web directory in Fig. 3 (left). Notice that in this website, unlike PVS, each level does not correspond to an information category (e.g., state or party). For instance, a hyperlink labeled 'soccer' resides at levels two and three.

To capture input expansion in non-levelwise sites, we use the concept of a *negative* FD: $x \rightarrow \neg\ y$, that holds when *none* of the paths through the site

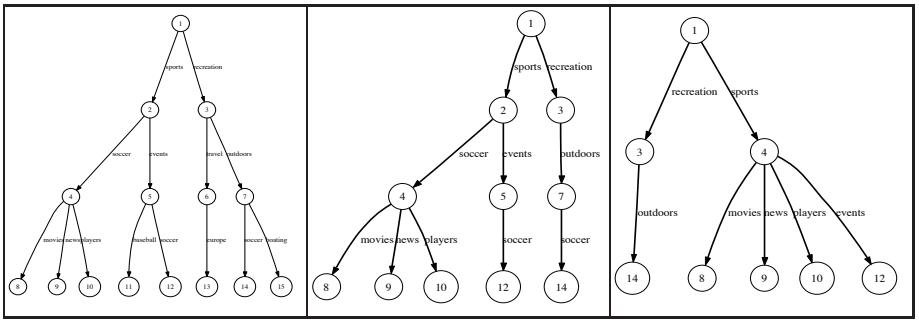

Fig. 3. (left) Hypothetical hierarchical web directory with characteristics similar to those in Yahoo!. (right and center) Customized versions of (left) wrt 'soccer'.

containing x also contain y. Some intuitive negative FDs in PVS are, 'Democrat $\rightarrow$ $\neg$ Republican', 'House $\rightarrow \neg$ Senate', and 'Virginia $\rightarrow \neg$ Ohio'. Notice that any negative FD $x \rightarrow \neg\ y$ implies $y \rightarrow \neg\ x$. When the user communicates 'Senate' out-of-turn, we can partially evaluate wrt to `Senate=true` and `House=false`. The reader will notice that negative FDs in a levelwise site involve only the labels of hyperlinks at the same level. However this is not the case in non-levelwise sites, thus providing the motivation for negative FDs. For instance, the following are some negative FDs that hold in the site illustrated in Fig. 3 (left): 'sports $\rightarrow \neg$ {boating, europe, outdoors, recreation, travel}' and 'soccer $\rightarrow \neg$ {baseball, boating, europe, travel}'. Thus, when a user says 'soccer' out-of-turn in Fig. 3 (left), we can partially evaluate the program in Fig. 4 (left) wrt `soccer=true` and `baseball`, `boating`, `europe`, and `travel` set to `false` which yields the program in Fig. 4 (right) which models Fig. 3 (right).

Notice that there are no salient structural properties of websites that ultimately influence the characteristics of their FDs (e.g., only FDs, only negative FDs, or a mixture of the two). The type of each FD currently satisfied by a site is dependent only on the current relationships between the co-occurrence of terms (labeling hyperlinks) on paths through the site (term y co-occurs on all/no paths containing term x). There are ways alternate to path containment in which terms can co-occur (e.g., two terms co-occur if the distinct paths which contain them lead to the same leaf vertex, i.e., the terms are used to index the same page) and using these criteria lead to additional types of FDs [17]. Further, notice the t threshold in our generalized notion of an FD defines the type of FD (t=0 specifies a negative FD and t=100 indicates an FD) for a particular co-occurrence criteria.

The distinction between levelwise and non-levelwise sites encouraged us to study the properties of web hierarchies to discern which program transformations are applicable to certain types of hierarchies. This analysis led to our development of a partial order of graph-theoretic classes of hierarchical hypermedia [17] which formally characterize websites by the relationships among the terms modeling the site's hyperlink labels. This ordering helps connect our work

```
 1 if (sports)          if (sports)        if (sports)
 2   if (soccer)          if (soccer)
 3     if (movies)          if (movies)        if (movies)
 4       page = 8;            page = 8;          page = 8;
 5     if (news)            if (news)          if (news)
 6       page = 9;            page = 9;          page = 9;
 7     if (players)         if (players)       if (players)
 8       page = 10;           page = 10;         page = 10;
 9   if (events)          if (events)        if (events)
10     if (baseball);
11       page = 11;
12     if (soccer);         if (soccer)
13       page = 12;           page = 12;         page = 12;
14 if (recreation)     if (recreation)    if (recreation)
15   if (travel)
16     if (europe)
17       page = 13;
18   if (outdoors)        if (outdoors)      if (outdoors)
19     if (soccer)          if (soccer)
20       page = 14;           page = 14;         page = 14;
21     if (boating)
22       page = 15;
```

Fig. 4. (left) Programmatic representation of the website modeled by Fig. 3 (left). (center) Representation of the site in Fig. 3 (center) resulting from slicing (left) wrt `music`. (right) Representation of site in Fig. 3 (right) resulting from partially evaluating (center) wrt only `soccer=true`.

to the hypermedia and interactive visualization community who have developed a similar taxonomy [18]. We refer the reader to [17] for a discussion of the formal details of the classes, detecting them, and proofs of their properties.

4.3 A General Program Transformation: Program Slicing

Even though we were able to generalize the support for out-of-turn interaction to non-levelwise sites, we still wanted to develop a general purpose program transformation technique than simply applying a combination of FDs and partial evaluation. One reason for this is that often only a subset of the terms in the consequent of a negative FD employed are necessary for partial evaluation. For example, partially evaluating the program in Fig. 4 (left) wrt `sports=true` and `recreation=false` results in the same program as would a partial evaluation wrt to `sports=true` and `boating`, `europe`, `outdoors`, `recreation`, and `travel` set to `false`. This encouraged us to study non-semantic-persevering transformations (ref. Table 1), such as program slicing [7], to generalize our approach to different forms of hierarchical hypermedia in a *single* framework. The idea involves slicing a program to retain only those sequences annotated with the user's out-of-turn input [19]. Program slicing [7] is a theoretical operation used to ex-

	Syntax-preserving	Semantic-preserving
Partial evaluation	×	√
Program slicing	√	×

Table 1. Comparison of partial evaluation and program slicing along a syntax- vs. semantic-preserving dichotomy.

tract statements that affect (or are affected by) the computation of a variable of interest at a point of interest from a program. There are several varieties of slicing; backward and forward slicing are the two most relevant for our purposes. Slicing has been predominately applied to problems in software engineering such as debugging and reverse engineering [7]. However, it has been applied applied to web application development [11]. Our use of it here helps relate it to interactive IR.

When the user says 'soccer' out-of-turn in Fig. 3 (left) we forward slice the program in Fig. 4 (left) wrt the `soccer` variable at all program points. This leads to the (`page` assignment) statements at lines 4, 6, 8, 13, and 20 from which we backward slice the program. The result is a representation of the site containing only paths involving hyperlinks labeled 'soccer' leading to leaf webpages containing information about soccer (ref. Fig. 4, center). Finally, we partially evaluate the program wrt the variable modeling the user's input ('soccer') statically set to `true` thereby removing all expressions involving it, since it has now been supplied. This results in a program modeling the new site (ref. Fig. 4, right) from which an actual site can be recreated. This program transformation technique generalizes out-of-turn interaction to all of the classes of hierarchical hypermedia that we identified.

4.4 A Duality in Uses of Program Slicing

The instructive nature of our use of generative programming suggested that an attempt to compute web FDs via program transformation might reveal more insight. We developed a technique which uses program slicing to mine web FDs from a programmatic representation of a website. We refer the reader to [17] for the details of the program transformation technique and rather focus on its implications here. We use partial evaluation and program slicing as pruning operators. There is a tradeoff between these two program transformations in the context interactive IR. Specifically, one can think of program slicing as a transformation for

1. directly pruning a website (as illustrated above in Fig. 4), *or*
2. extracting information (i.e., FDs) about *what* to prune from a site and then using this information with partial evaluation to conduct the same site pruning as in (1).

Studying this duality reveals that there might be simpler or more effective methods for realizing out-of-turn interaction with instances of specialized classes in our taxonomy, akin to that illustrated in section 4.2 for levelwise sites.

Towards a Taxonomy of Program Transformations for Interactive IR
Thus far, we have seen that our generative approach using only partial evaluation works for only levelwise sites, the most specific class in our partial order. In addition, we have illustrated an alternate program transformation technique, based on slicing, to realize out-of-turn interaction in all of the classes of hierarchical websites we identified. This suggests that additional specialized transformation techniques might exist for classes in our lattice between the most general and most specific class. Developing a mapping between classes of hierarchical hypermedia and generative techniques for interacting with them moves us closer to a generative programming model for interactive IR.

This section has described many insights and made several connections. To recap, we have seen that

1. our simplification rules, involving program transformers, led to the idea of a web FD,
2. web FDs led to a new way to conduct query expansion on the web and, ultimately, approximate interactive IR,
3. the application of partial evaluation as a pruning operator led to classes of hierarchical hypermedia,
4. supporting out-of-turn interaction with instances of the classes led to two new generative approaches: one involving a combination of FDs and partial evaluation and another involving program slicing which generalizes out-of-turn interaction to each class,
5. the two new generative approaches and a method to mine FDs with slicing led to a duality in uses of program slicing, and
6. we are optimistic that this duality will lead to a taxonomy of program transformations for interactive IR.

Overall, this generative programming thread resulted in an improved understanding of web taxonomies and new research issues.

4.5 New Research Issues for Web Taxonomies

Many large web taxonomies, such as Yahoo! and the Open Directory Project at `dmoz.org`, are modeled as a DAG (Directed Acyclic Graph), owing to the presence of symbolic links. A symbolic link is a special type of hyperlink which makes a directed connection from a webpage along one path through a website to a page along another path. One obvious use of symbolic links is multiclassification. For example, information about music is classified under both the arts and computers categories in Fig. 3 (left). Rather than classify information under more than one category, a designer might classify it under only one category, but include a symbolic link from one category to another (e.g., from the arts sub-tree to the computers sub-tree, or vice versa) to give the user the illusion that the item is classified in both categories. Representing a website modeled as a DAG using a program is challenging and requires the use of unconditional branches (e.g., `goto`s) or functions to factor common branches. This viewpoint

leads us to associate a symbolic link with a kludge for out-of-turn interaction. We hypothesize that designers include symbolic links to address the fact that users do not have facilities to interact out-of-turn. Testing this hypothesis suggests that we should mine the uses of symbolic links on the web to identify the typical contexts in they are employed. Replacing symbolic links with new interaction techniques which more naturally support users' information-seeking activities relieves the designer from having to anticipate where and how to include symbolic links in a taxonomy to accommodate users with diverse informational goals.

Lastly, notice that we might generalize our definition of an FD even further to capture and expose the various relationships that might exist between the terms that label hyperlinks in websites. For instance, rather than saying that the $x \rightarrow y$ FD holds if $t\%$ of the paths through the site involving x also involve y, we might say the $x \rightarrow y$ FD holds if $M\ (x,\ y) \geqslant t$, where M is a term similarity metric from IR (e.g., cosine or Jaccard's [20]) and t is a threshold. Thus, another research issue that this thread revealed involves experimentation to identify the metrics and threshold values that are appropriate for a given (class of website, information-seeking goal) pair to be supported.

5 New Web Interaction Techniques and Interfaces

5.1 User Interfaces for Approximate Interactive IR

Approximate retrieval in a web taxonomy, introduced above, is important because, by exposing term relationships (similarities), it can help a user assimilate the underlying domain by dependency exploration. It also can reveal hidden aspects of the domain. For example, a user that communicates 'Senate Senior' and observes that it expands to 'Senate Senior Democrat' might conclude that the Senior leadership in the Senate is largely Democratic, an inference difficult to make simply by browsing the site. Such approximate interactive IR suggests that we might expand a query in *real-time* for the user or permit the user to set the expansion threshold (from no expansion to as much expansion as possible) and *dynamically* observe the links that are removed or added as the user, e.g., moves a slider bar UI widget to dynamically adjust the threshold.

5.2 Dialog Continuations

Our footing on the generative landscape suggested investigating addition techniques from programming languages employed in program transformations, such as continuations. We then used continuations to design a new web interaction primitive, *dialog continuations*, intended to address the destructive nature of program transformations on interaction. To support procedural tasks, we must allow for new subdialogs to be dynamically invoked at the behest of the user, who also determines any partial input that might be applicable at that point. To achieve this functionality, we explicitly manipulate dialog continuations, borrowing an important notion from the programming languages literature [8]. A continuation indicates a 'promise to do something' and summarizes the amount

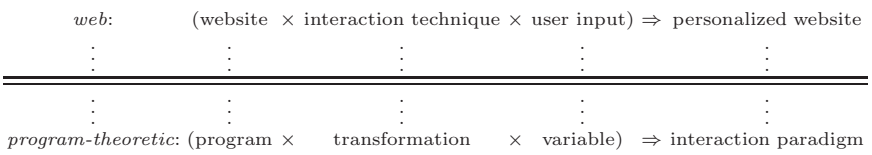

Fig. 5. The connection between the web and program-theoretic domains.

of work remaining at a point of execution in a program. While all languages employ continuations internally, only some (e.g., Scheme) allow the user to explicit manipulate and reason about them. To cascade one dialog onto another, we essentially replace the current continuation with a fresh dialog that has been partially evaluated with the user's chosen input, and jump to this dialog. This allows the user to both abandon a given line of conversation (since the requisite information has been obtained) and find themselves in the middle of another line of inquiry.

We have implemented a real-time query expansion interface and various dialog continuation interfaces for users to interactively explore the PVS data. They are available for demonstration from our project webpage at `http://oot.cps.udayton.edu`.

6 Discussion

We have described the insight gained by the virtue of a generative programming lens for interactive IR. The central theme of our generative approach is to pose interactive information retrieval as the application of a program transformations to a programmatic representation of a website based on partial user input (ref. Fig 5, bottom). The creativity in our work (ref. Fig. 5) arises from relating concepts in the web domain (e.g., sites, interactions) to notions in the program-theoretic domain (e.g., programs, transformations). An additional opportunity for creativity involves varying the (program, transformation) pair to achieve a desired form of interaction. The predominate form of interaction discussed in this article is out-of-turn interaction.

The generative techniques showcased here can be implemented with many software tools or programming languages. Our implementation employs PHP for the transformation, generative, and mining components and XUL, SALT, and JavaScript for the user interaction interfaces. Our generative approach has not only been instructive, but also has led to a simple implementation strategy. We implemented the entire framework using less then 1000 lines of code, where the constituent components each occupy approximately equal amounts of code and are cleanly factored. The framework contains no code specific to a targeted website. The representation is the only component which contains site-specific information and is supported as plug-in basis (or simply stored on the original website's server). For further implementation details, including caching and sessioning, see [4]. We have applied these techniques and our framework to several websites: (i) GAMS (Guide to Available Mathematical Software) at

gams.nist.gov, (ii) Project Vote Smart at vote-smart.org, (iii) CITIDEL (Computing and Information Technology Interactive Digital Educational Library) at citidel.org [21], (iv) the Open Directory Project at dmoz.org, and (v) the Online Virginia Tech Timetables of Classes accessible through vt.edu. We refer the reader to [3, 4, 17] for the details of these case studies.

In summary, the insight exposed by our use of generative programming has (i) helped us connect our work to other communities, (ii) driven the development of new concepts, and (iii) led to new research issues. In particular, we connected our work to the discourse analysis (dialog) and hypermedia/visualization communities. The concept of an FD on the web, while simple, suggested a new way to do query expansion on the web and led to approximate retrieval in large web taxonomies. In addition, borrowing notions like continuations from programming languages led to new, richer, and more conversational, ways of interacting with websites. We intend to continue to investigate the use of generative programming for interactive IR. For example, we might use a language's support for *reflection* to permit the user to dynamically query the program for the choices that are still unspecified as a way of enquiring 'what may I say at this point in the dialog?'

Another compelling line of future work entails investigating what the axiomatic semantics of a program modeling a website imply about the forms of interaction supported by the site. In other words, what are the web interaction analogs to the axiomatic semantics of a program modeling web interaction? An example of a simple (and obvious) interaction axiom which can be inferred from the program is 'no customer shall reach the thank you page without first paying for the items in their shopping cart.' We are optimistic that this work will help us automatically reason about interacting with a site from program axioms. We anticipate such automated reasoning to become more important with the growth of initiatives advocating for more automation, such as the *semantic web* [22] which aims to lift the communication paradigm of the web from human-to-computer to computer-to-computer. For this reason, we believe that the generative approach espoused here is especially timely. The long-term goal of this work is to use the insight detailed here to develop general, but automated, models for the design of interactive (and responsive) websites.

References

1. Marchionini, G.: Information Seeking in Electronic Environments. Cambridge Series on Human-Computer Interaction. Cambridge University Press (1997)
2. Allen, J.F., Guinn, C.I., Horvitz, E.: Mixed-Initiative Interaction. IEEE Intelligent Systems Vol. 14 (1999) pp. 14–23
3. Perugini, S., Ramakrishnan, N.: Personalizing Web Sites with Mixed-Initiative Interaction. IEEE IT Professional Vol. 5 (2003) pp. 9–15
4. Narayan, M., Williams, C., Perugini, S., Ramakrishnan, N.: Staging Transformations for Multimodal Web Interaction Management. In: Proceedings of the Thirteenth ACM International World Wide Web Conference (WWW), New York, NY (2004) pp. 212–223
5. Jones, N.D.: An Introduction to Partial Evaluation. ACM Computing Surveys Vol. 28 (1996) pp. 480–503

6. Ullman, J.: Elements of ML Programming. Second edn. Prentice-Hall (1997)
7. Tip, F.: A Survey of Program Slicing Techniques. Journal of Programming Languages Vol. 3 (1995) pp. 121–189
8. Friedman, D.P., Wand, M., Haynes, C.T.: Essentials of Programming Languages. Second edn. MIT Press (2001)
9. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)
10. Belkin, N.J., Cool, C., Stein, A., Thiel, U.: Cases, Scripts, and Information-Seeking Strategies: On the Design of Interactive Information Retrieval Systems. Expert Systems with Applications Vol. 9 (1995) pp. 379–395
11. Ricca, F., Tonella, P.: Web Application Slicing. In: Proceedings of the International Conference on Software Maintenance (ICSM), Florence, Italy (2001) pp. 148–157
12. Ricca, F., Tonella, P., Baxter, I.D.: Restructuring Web Applications via Transformation Rules. In: Proceedings of the First International Workshop on Source Code Analysis and Manipulation (SCAM), Florence, Italy (2001) pp. 150–160
13. Graunke, P., Findler, R., Krishnamurthi, S., Felleisen, M.: Automatically Restructuring Programs for the Web. In: Proceedings of the Sixteenth IEEE International Conference on Automated Software Engineering (ASE), San Diego, CA (2001) pp. 211–222
14. Queinnec, C.: The Influence of Browsers on Evaluators or, Continuations to Program Web Servers. In: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP), Montreal, Canada (2000) pp. 23–33
15. Quan, D., Huynh, D., Karger, D.R., Miller, R.: User Interface Continuations. In: Proceedings of the Sixteenth Annual ACM Symposium on User Interface Software and Technology (UIST), Vancouver, Canada (2003) pp. 145–148
16. Agrawal, R., Imielinski, T., Swami, A.N.: Mining Association Rules between Sets of Items in Large Databases. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD), Washington, DC (1993) pp. 207–216
17. Perugini, S.: Program Transformations for Information Personalization. Ph.D. dissertation, Department of Computer Science, Virginia Tech (2004) Available in the Virginia Tech ETD collection at http://scholar.lib.vt.edu/theses/available/etd-06252004-162449/. US Copyright Office Registration Number TX 6-040-012.
18. McGuffin, M.J., m. c. schraefel: A Comparison of Hyperstructures: Zzstructures, mSpaces, and Polyarchies. In: Proceedings of the Fifteenth ACM Conference on Hypertext and Hypermedia (HT), Santa Cruz, CA (2004) pp. 153–162
19. Perugini, S., Ramakrishnan, N.: Personalization by Program Slicing. Journal of Object Technology Vol. 4 (2005) pp. 5–11 Special issue: *Sixth ACM GPCE Young Researchers Workshop*, Vancouver, Canada, October 2004.
20. Srehl, A., Ghosh, J., Mooney, R.: Impact of Similiarity Measures on Web-page Clustering. In: Proceedings of the AAAI Worshop of Artificial Intelligence for Web Search, Austin, TX (2000) pp. 58–64
21. Perugini, S., McDevitt, K., Richardson, R., Pérez-Quiñones, M.A., Shen, R., Ramakrishnan, N., Williams, C., Fox, E.A.: Enhancing Usability in CITIDEL: Multimodal, Multilingual, and Interactive Visualization Interfaces. In: Proceedings of the Fourth ACM/IEEE Joint Conference on Digital Libraries (JCDL), Tucson, AZ (2004) pp. 315–324
22. Despeyroux, T.: Practical Semantic Analysis of Web sites and Documents. In: Proceedings of the Thirteenth ACM International World Wide Web Conference (WWW), New York, NY (2004) pp. 685–693