Program Transformations for Information Personalization

Saverio Perugini

Department of Computer Science
University of Dayton
300 College Park, Dayton, OH 45469–2160, USA

Naren Ramakrishnan

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061–0106, USA

Abstract

Personalization constitutes the mechanisms necessary to automatically customize information content, structure, and presentation to the end-user to reduce information overload. Unlike traditional approaches to personalization, the central theme of our approach is to model a website as a program and conduct website transformation for personalization by program transformation (e.g., partial evaluation, program slicing). The goal of this paper is study personalization through a program transformation lens, and develop a formal model, based on program transformations, for personalized interaction with hierarchical hypermedia. The specific research issues addressed involve identifying and developing program representations and transformations suitable for classes of hierarchical hypermedia, and providing supplemental interactions for improving the personalized experience. The primary form of personalization discussed is *out-of-turn interaction* – a technique which empowers a user navigating a hierarchical website to postpone clicking on any of the hyperlinks presented on the current page and, instead, communicate the

Email addresses: `saverio@udayton.edu` (Saverio Perugini), `naren@cs.vt.edu` (Naren Ramakrishnan)

URL: `http://academic.udayton.edu/SaverioPerugini` (Saverio Perugini), `http://people.cs.vt.edu/~naren` (Naren Ramakrishnan)

label of a hyperlink nested deeper in the hierarchy. When the user supplies out-of-turn input we personalize the hierarchy to reflect the user's informational need. While viewing a website as a program and site transformation as program transformation is non-traditional, it offers a new way of thinking about personalized interaction, especially with hierarchical hypermedia. Our use of program transformations casts personalization in a formal setting and provides a systematic and implementation-neutral approach to designing systems. Moreover, this approach helped connect our work to human-computer dialog management and, in particular, mixed-initiative interaction. Putting personalized web interaction on a fundamentally different landscape gave birth to this new line of research. Relating concepts in the web domain (e.g., sites, interactions) to notions in the program-theoretic domain (e.g., programs, transformations) constitutes the creativity in this work.

Key words: hierarchical hypermedia, information personalization, navigation, out-of-turn interaction, program transformations, partial evaluation, program slicing, web interaction, web mining, website transformation

---

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

Sir William Lawrence Bragg,
the youngest-ever recipient of the Nobel Prize.

1. Introduction

Information personalization constitutes the mechanisms necessary to automatically customize information content, structure, and presentation to the end-user to reduce information overload (Perugini and Ramakrishnan, 2003a). Personalization technologies are now ubiquitous on the web and critical to retaining customers (e.g., eBay, Amazon).

Our view of personalization is oriented toward personalizing user *interaction.* Specifically, we have developed an interaction technique which empowers a user navigating a hierarchical website to postpone clicking on any of the hyperlinks presented on the current page and, instead, communicate the label of a hyperlink nested deeper in the hierarchy. We call this technique *out-of-turn interaction* and when the user supplies out-of-turn input (i.e., a hyperlink label) we re-organize (or, in other words, personalize) the hierarchy to reflect the user's informational need.

Consider a user, interacting with an automobile website, such as Edmunds, interested in manufacturers offering hybrid automobiles. If the site's hierarchical structure requires the user to select a manufacturer at the top level, make at the following level, and so on, then to fulfill the information-seeking goal, this user would need to drill-down through each manufacturer and manually aggregate all hybrid automobiles discovered at the lower levels of the site. However, using out-of-turn interaction, this user could say 'hybrid' at the top level of the site and in response the system would prune out all manufacturer hyperlinks on the root page which do not lead to hybrid automobiles and, therefore, only present hyperlink representing manufacturers which offers hybrid models. With this set of reduced manufacturers the user has the option of browsing (i.e., clicking on one of the presented hyperlinks) or, again, interacting out-of-turn (e.g., by saying 'manual transmission').

Out-of-turn interaction permits the user to circumvent any intended flows of navigation hardwired into a hyperlink structure by the designer and, in this manner, flexibly reconciles any mismatch between the site's one-size-fits-all organization and the user's model of information seeking. Out-of-turn input can be communicated to a site either through text using a browser toolbar plugin (Perugini and Ramakrishnan, 2003b) or through speech using a voice user interface (Narayan et al., 2004).

Unlike traditional approaches to personalization, the central theme of our approach is to model information-seeking interactions with hierarchical hypermedia explicitly in a programmatic representation and use program transformations (e.g., partial evaluation, program slicing) to stage the interaction (Perugini and Ramakrishnan, 2005). A *program transformation* is an automatic, closed operation mapping one program to another. Converting a program which computes x^n to one which computes x^2 is a simple example of a transformation (in this case, partial evaluation). *Program slicing* (Binkley and Gallagher, 1996) is a program transformation used to extract statements which may affect or be affected by the values of variables from a program.

A website such as the *Yahoo!* directory may be viewed as a DAG, with vertices representing webpages, edges representing hyperlinks, edge labels representing hyperlink labels or search terms, and leaves representing content pages (destinations) (e.g., see Fig. 1). We support the user in experiencing a personalized traversal of such a website by permitting her to enter a search term out-of-turn and adjusting the graph accordingly, e.g., by retaining only the subgraph leading to leaves which have an occurrence of the search term on a path from the root to each leaf (e.g., see Fig. 2). A DAG may also be represented by a program, where each search term corresponds to a boolean variable in a branch in a nested conditional representation (e.g., see Table 4). In that representation, the out-of-turn adjustment described above is modeled by slicing the program. Thus, the essence of this paper is an equivalence between rooted DAGs and programs, such that some operations of interest on a DAG (such as adjustment to out-of-turn search terms) correspond to operations of interest on a program (such as slicing).

In summary, the central theme of our research is to pose website personalization and, particularly, website transformation, as the application of a program transformation technique to a programmatic representation of interaction based on (often partial) user input. This approach offers a new way of thinking about personalized interaction, especially with hierarchical hypermedia. Decoupling the logic into a program (representation, transformation) pair[1] provides a clean separation of concerns and allows us to personalize a hierarchy to individual users without explicitly enumerating a specialized hierarchy (or building a user model) for each individual user. It also fosters the attractive possibility of exploring alternate representations and transformations and studying the resulting forms of personalization enabled. The creativity in this research arises from relating concepts in the web domain (e.g., sites, links) to notions in the program-theoretic domain (e.g., programs, transformations) (see Table 1).

1.1. Objectives

We have built a software framework based on the theoretical ideas presented in this paper (Narayan et al., 2004) and have conducted human-computer interaction studies with users to evaluate specific systems designed with it (Perugini, 2004, Ch. 6) (Perugini et al., 2007). The goal of this paper is study personalization through a program transformation lens, and develop a formal model, based on program transformations, for personalized interaction with hierarchical hypermedia. The primary form of personalization

[1]The representation in a program (representation, transformation) pair specifies how to model a website as a program, such as the nested conditionals representation used in the programs shown in Table 4, while the transformation is a program transformation, i.e., a closed, source-to-source operation mapping a program to another program, such as partial evaluation or program slicing.

Table 1: Analogs between the web interaction and program-theoretic domains.

Web interaction:	Transformation(··· Transformation	(Website,	Hyperlink label),	···, Hyperlink label)	⇒	Personalized website
⋮	⋮	⋮	⋮	⋮		⋮
⋮	⋮	⋮	⋮	⋮		⋮
Program-theoretic:	Transformation(··· Transformation	(Program,	Program construct),	···, Program construct)	⇒	Specialized program

discussed here is out-of-turn interaction. The objectives of this paper are to

1. develop graph-theoretic interpretations of out-of-turn interaction with a general class of websites,
2. illustrate how these interpretations can be supported by a program (representation, transformation) pair (called a *model*), often involving program slicing,
3. evaluate the soundness and completeness (as well as other properties) of a model wrt a target interaction paradigm (e.g., browsing or out-of-turn interaction),
4. identify a partial order of classes of hierarchical hypermedia and explain its implications on the program transformation approach to personalization,
5. introduce web functional dependencies and describe how they can be mined from websites and used for automatic query expansion to personalize the user experience further,
6. illustrate program transformation techniques based on program slicing to mine web functional dependencies,
7. demonstrate that an alternate program transformation technique, which employs web functional dependencies, can achieve the same effect as the original transformation technique,
8. develop specialized program transformation techniques for some specific classes of hierarchical hypermedia, and
9. demonstrate how the program transformation formalism can be used to support supplementary personalized interactions.

1.2. Research Methodology

When a user says something out-of-turn, we ask: what can be reasonably pruned out of the website? Answers to this question lead to interpretations of out-of-turn interaction, e.g., when a user says something out-of-turn,

Table 2: Our research methodology.

$$\begin{tikzcd}[row sep=small]
\text{Develop a graph-theoretic definition of an interpretation of personalized interaction.} \arrow[d] \\
\text{Model interaction with a hierarchical website as a program (never to be executed, but only to be transformed).} \arrow[d, shift right] \\
\text{[design a mapping from user input to program constructs to capture requirements]} \arrow[u, shift right] \arrow[d, shift right] \\
\text{Develop a program transformation technique capable of realizing the interpretation.} \arrow[u, shift right] \arrow[d] \\
\text{Evaluate the model.} \arrow[d] \\
\text{Study the enabled personalized interaction with users.} \arrow[uuuuu, bend left]
\end{tikzcd}$$

1. (a) first identify leaf webpages reachable by a path involving a hyperlink labeled with the out-of-turn input, and
   (b) prune all paths through the site that do not lead to any of these leaf pages.
2. prune all paths through the site which do not involve a hyperlink labeled with the out-of-turn input.

Both interpretations assume that the input supplied by the user is legal. An interaction using illegal input may either be undefined or cause an error. Interpretation 2 entails interpretation 1 because every path retained by interpretation 2 is retained under interpretation 1. In other words, interpretation 2 prunes all paths pruned under interpretation 1, but the converse does not hold. Next we ask: how can we support (i.e., model and realize) the interpretation of out-of-turn interaction using program-theoretic principles.

This suggests an iterative process, illustrated in Table 2, of developing a programmatic representation of interaction with an instance of hierarchical hypermedia and developing a program transformation technique, often involving a composition of program transformations, capable of supporting the desired personalized interactions from the model. Moreover, we must design a mapping from user requests (often partial input) to program constructs

Table 3: Graph-theoretic constructs and web analogs.

Graph-theoretic construct	**Web analog**
Graph	Website
Vertex	Webpage
Edge	Hyperlink
Edge-label	Hyperlink label
Root	Homepage

(often variables) to direct the transformation. We next evaluate the model by computing various metrics. We evaluate the personalized interaction enabled by conducting studies with users (Perugini, 2004, Ch. 6) (Perugini et al., 2007) which often reveal insights into new interpretations, interaction paradigms, and interaction techniques and, thus, help close the loop.

While some parts of this paper have been previously reported upon by the authors (Narayan et al., 2004; Perugini and Ramakrishnan, 2005), the present paper builds upon these foundations to develop a unifying theory for personalization using program transformations. Where necessary, key results from the above papers (e.g., Table 1, taken from (Perugini and Ramakrishnan, 2005); Table 5, taken from (Narayan et al., 2004); Table 6, taken from (Perugini and Ramakrishnan, 2005), and Section 8, taken from (Perugini and Ramakrishnan, 2005)) are reported to ensure that the discussion here is self-contained.

2. Graph-theoretic View of Personalized Interaction

We begin by developing syntactic notions from graph theory and progressively attach web interaction semantics to develop a theory of representing and reasoning about interaction with hierarchical hypermedia.

2.1. Syntactic and Semantic Notions

Fig. 1 illustrates a DAG model of a hierarchical website with characteristics similar to web directories such as *Yahoo!* at `http://dir.yahoo.com` or the *Open Directory Project* (ODP) at `http://dmoz.org`. Table 3 is an abridged mapping from graph-theoretic notions to web analogs. Edges help model paths through a website a user follows to access leaf vertices. Leaf vertices model leaf webpages which contain content. We refer to a leaf

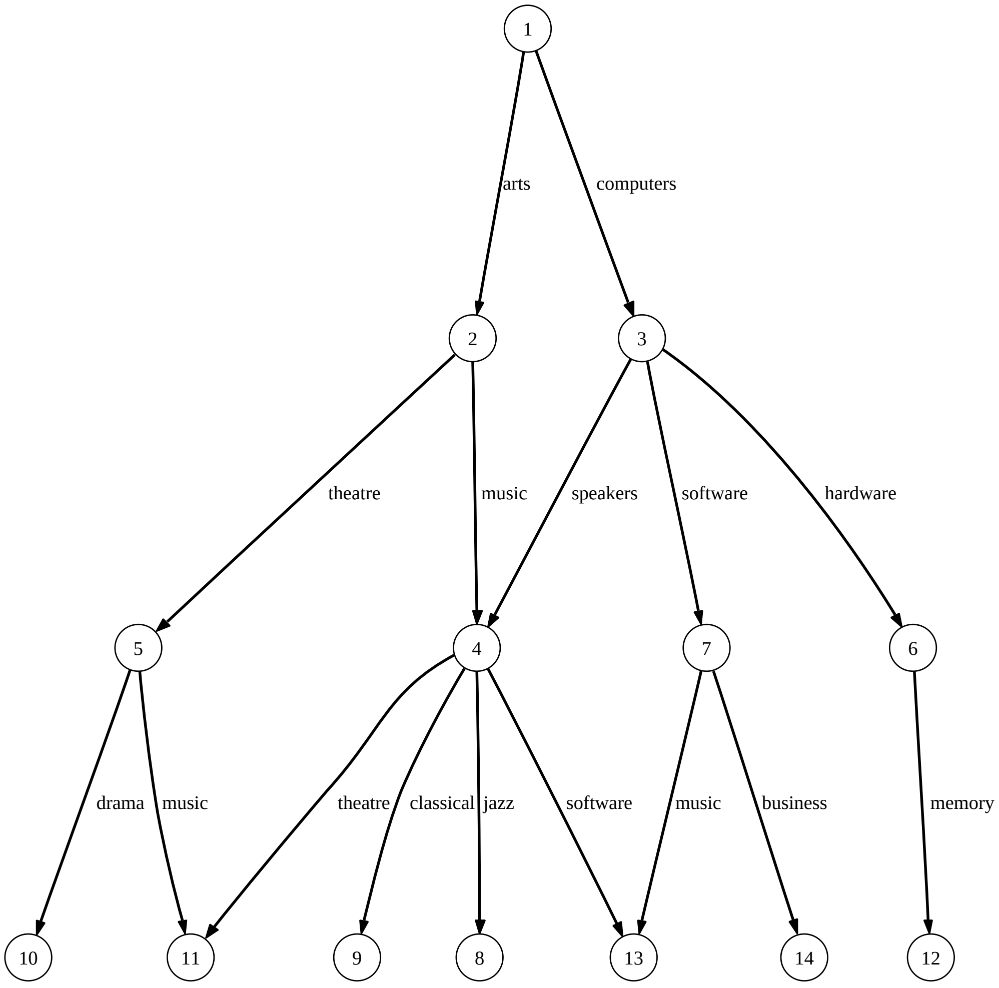


Figure 1: Example of a DAG model of a hypothetical hierarchical website with characteristics similar to those in the *Yahoo!* directory

content page as *terminal information* and the terms therein as *units of terminal information*. Edge-labels, which we refer to as *structural information*, model hyperlink labels or, in other words, choices made by a navigator en route to a leaf. An edge-label, a *unit of structural information*, is therefore a *term of information-seeking* (simply a *term* hereafter) which a user may bring to bear upon information seeking. Structural information thus helps make distinctions among terminal information.

A set of terms is *complete* when it determines a particular leaf webpage; otherwise it is *partial*. An *interaction set* of a DAG D is the complete set of the terms along a path from the root of D to a leaf vertex of D. An interaction set constitutes complete information; any proper subset of it is partial information. An interaction set of D classifies a leaf vertex of D, but does not capture any order of the terms.

We now provide definitions which pertain to a user's interaction with a website. A term is *in-turn* information if it appears as a hyperlink label on the user's current webpage and is, thus, currently solicited by the system. On the other hand, a term is *out-of-turn* information if it represents a hyperlink label nested somewhere deeper in the site and is, thus, currently unsolicited from the system, but relevant to information seeking. Each term from D which is not in-turn information is out-of-turn information.

Several partial orders can be defined over an interaction set wrt the time at which the user communicates the term to the system, called *arrival time*. When a user clicks on a hyperlink, she implicitly communicates the hyperlink label to the underlying system. For instance, when a user clicks on the hyperlink labeled 'arts' followed by that labeled 'music,' she communicates the $\prec$arts, music$\succ$ terms, in that order. Similarly, when the user supplies out-of-turn input (using a textual or speech modality), he is communicating terms to the system. These partial orders can be summarized with partially ordered sets or posets. Each linear extension of such a poset is a total order called an *interaction sequence*. A *browsing interaction sequence* of D is a total order on an interaction set of D wrt the parenthood relation of D. An *out-of-turn interaction sequence* of D is a total order on an interaction set of D wrt the arrival time relation implied by out-of-turn interaction. Interestingly, both interpretations of out-of-turn interaction introduced above imply the same arrival time relation – a partial order containing only the reflexive tuples of terms from the interaction set. In other words, none of the terms from the interaction set are required to be ordered. The linear extensions of the posets associated with these partial orders are out-of-turn interaction sequences.

An *interaction paradigm* $\mathcal{P}$ for D is given by the union of all linear extensions of posets defined over interaction sets of D. In other words, an interaction paradigm is a complete set of realizable interaction sequences from D wrt an interaction technique. When an edge-label labels more than one edge in a path from the root of D to a leaf vertex of D, it is advantageous to think of an interaction sequence as a finite effective enumeration of an interaction set of D, where the order of the terms in the enumeration corresponds to the arrival time relation afforded by the interaction technique wrt D. The browsing paradigm of D in Fig. 1 is $\{$≺arts, music, jazz≻, ≺arts, music, classical≻, ≺arts, music, theatre≻, …, ≺computers, hardware, memory≻$\}$. Likewise, an out-of-turn paradigm is

{≺arts, music, jazz≻, ,≺music, arts, jazz≻,
[the remaining 4 permutations of {arts, music, jazz}],
≺arts, music, classical≻, ≺music, arts, classical≻,
[the remaining 4 permutations of {arts, music, classical}],
…,
≺computers, hardware, memory≻, ≺hardware, computers, memory≻,
[the remaining 4 permutations of {computers, hardware, memory}]}.

While there can be only one browsing paradigm, there are multiple out-of-turn paradigms. Moreover, the browsing paradigm for a DAG D is a subset of any out-of-turn paradigm for D.

2.2. Support Terms and Tools

We use the symbol $\mathcal{D}$ to represent the universal set of DAGs, the symbol $\mathcal{T}$ to represent the universal set of terms, and $\mathcal{L}$ to denote the universal set of leaf webpages. Before we can expand this discussion to functions over $\mathcal{D} \times \mathcal{T}$ to realize the sequences of a particular interaction paradigm, we must develop some support terms and tools.

Sequencize is a total function $SQ : \mathcal{D} \rightarrow P(\mathcal{I})$ which given D returns the browsing paradigm of D. We use the symbol $\mathcal{I}$ to represent the universal set of interaction sequences. $P(\cdot)$ denotes the power set function.

Term extraction is a total function $TE : \mathcal{D} \rightarrow P(\mathcal{T})$ which given D returns the set of all unique terms in D. A *term-co-occurrence set* of D is a set $T \subseteq TE(D)$. Let the *level* of an edge-label in D be the depth of the source vertex of the edge it labels. If a given edge-label occurs multiple times in D, a level is associated with every occurrence. A *term-level set* of D is a term-co-occurrence set comprising all unique terms in D with the same level. *Term-level extraction* is a total function $TLE : (\mathcal{D} \times \mathcal{N}) \rightarrow P(TE(D))$ which

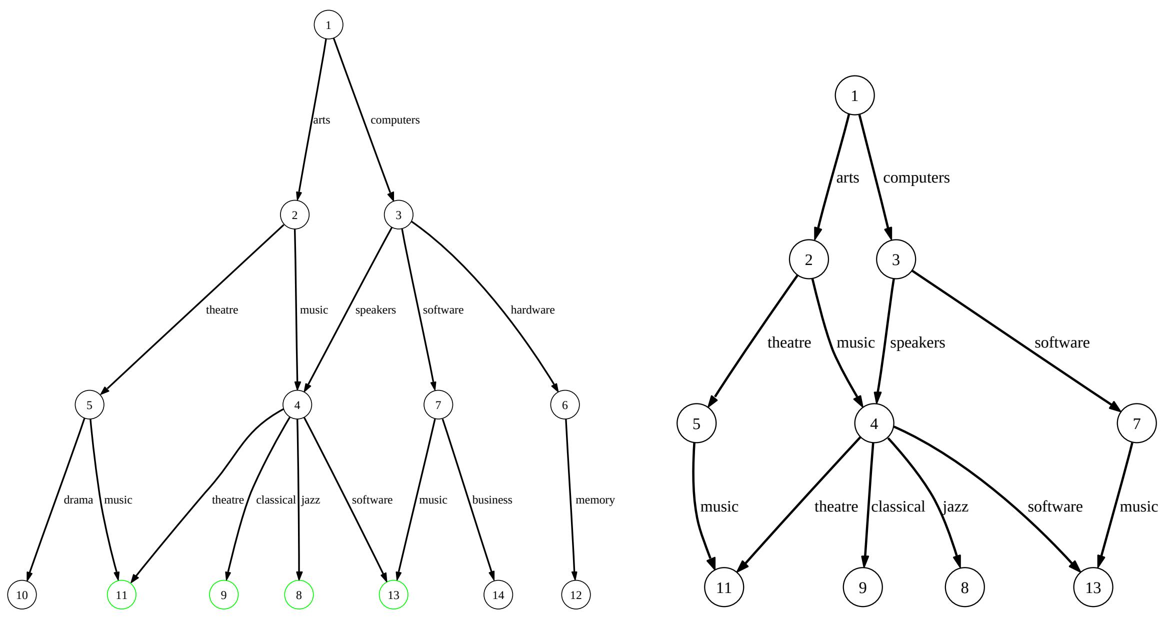


Figure 2: Illustration of forward-propagation (FP) followed by back-propagation (BP) on the DAG in Fig. 1. (left) Forward-propagation wrt the term ‘music,’ i.e., $FP(D,\ \text{music})$. (right) Back-propagation wrt the leaf vertices highlighted green (lighter in black and white rendering) in left, i.e., $BP(D,\ FP(D,\ \text{music}))$.

given D and a level $l(\geqslant 1) \in \mathcal{N} = \{1, 2, \ldots, M\}$ returns the set of all unique terms in D with level l, i.e., a term-level set. We use the variable M to represent the the maximum depth of D. If D represents the DAG in Fig. 1, $TLE(D,\ 2) = \{$theatre, music, speakers, software, hardware$\}$. In any DAG, $TLE(D,\ 1)$ returns the set of terms available to supply through browsing.

Get sequences is a partial function $GS : (\mathcal{T} \times P(\mathcal{I})) \rightarrow P(\mathcal{I})_{\perp}$ which given a term t and a set of interaction sequences I_S returns the set of all interaction sequences in I_S which each contain t as a member. The symbol $\perp$ denotes the partial nature of the function, i.e., the value of GS is undefined for some inputs. *Forward propagate* is a total function $FP : (\mathcal{D} \times T) \rightarrow P(\mathcal{L})$ which given D and a term $t \in T = TE(D)$ returns a set of leaf vertices L of D, where L contains each leaf vertex reachable from all paths of D containing an edge labeled t. *Collect results* is a total function $CR : \mathcal{D} \rightarrow P(\mathcal{L})$ which given D returns a set of all the leaf vertices in D. Collect results wrt the sub-DAG rooted at vertex 2 in Fig. 1 is the $\{8, 9, 10, 11, 13\}$ set of vertices. *Back propagate* is a total function $BP : (\mathcal{D} \times P(\mathcal{L})) \rightarrow \mathcal{D}$ which given D and a set of leaf vertices L returns a DAG $D^{'}$, where $D^{'}$ contains only browsing interaction sequences which classify the leaf vertices in L. Fig. 2 illustrates forward-propagation (left) on the DAG D in Fig. 1 followed by back-propagation (right) yielding $D^{'}$.

Notice that D also can be represented as a $|TE(D)| \times |CR(D)|$ term-document matrix, where rows correspond to terms (structural information, or edge-labels) and columns correspond to webpages (terminal information, or leaf vertices). However, such a matrix is insufficient to reconstruct D.

2.3. Interpretations of Out-of-turn Interaction

Prior to providing graph-theoretic interpretations for out-of-turn interaction, we formalize browsing over DAG models of websites. *Browse* is a partial function $B : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{D}_{\perp}$, which given D and a term $t \in TLE(D,\ 1)$ returns the sub-DAG rooted at the target vertex of the edge in D labeled with edge-label t whose source vertex is the root of D. If the DAG in Fig. 1 is D, $B(D,$ computers$)$ returns the sub-DAG rooted at vertex 3, which represents the result of a user clicking on the hyperlink labeled ‘computers.’

We formally cast the above two interpretations for out-of-turn interaction in graph-theoretic terms:

Interpretation 1 of out-of-turn interaction is a partial function $OOT_1 : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{D}_{\perp}$ which given D and a term $t \in TE(D)$ returns $D^{'}$. It is defined as

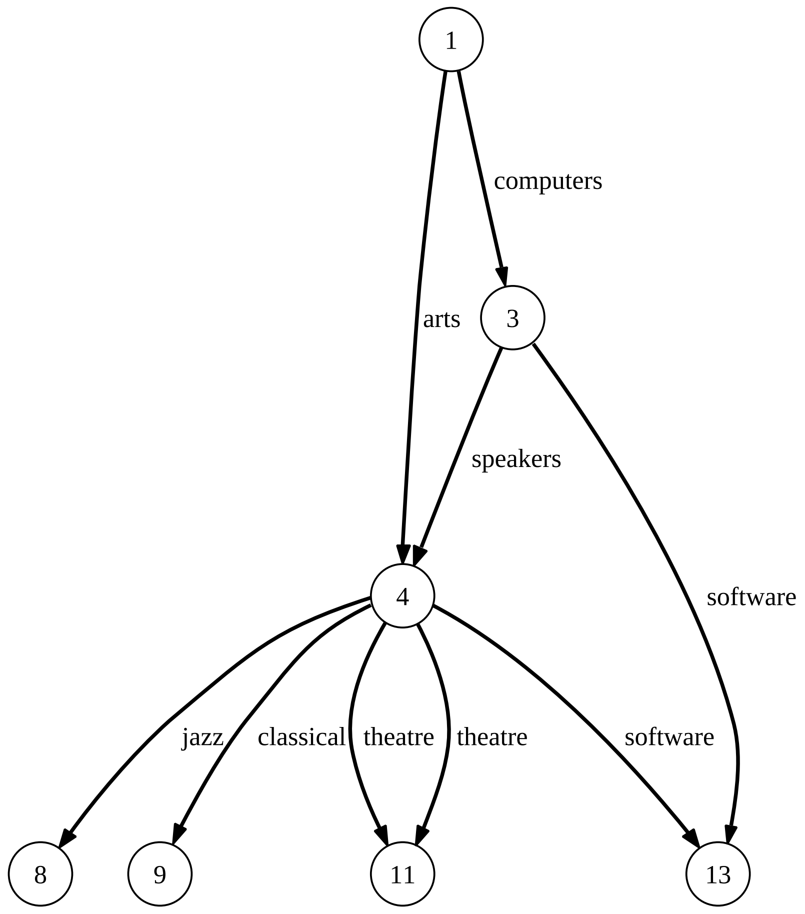
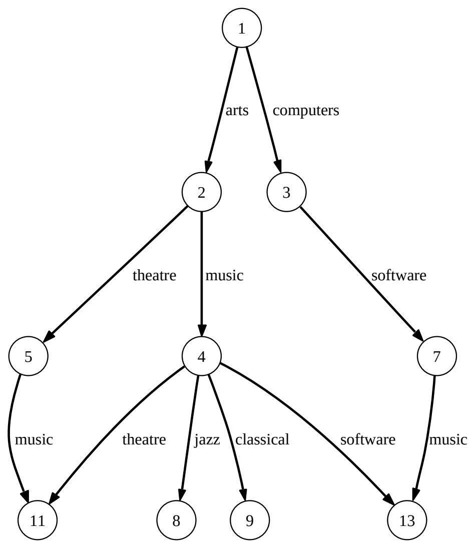


Figure 3: Result of interpretation 1 of out-of-turn interaction with the DAG D shown in Fig. 1 wrt the term 'music,' i.e., $OOT_1(D,\ \text{music})$. Alternatively, one can think of this DAG as the result of shrink edges with the DAG $D^{'}$ in Fig. 2 (right), i.e., $SE(D^{'},\ \text{music})$.



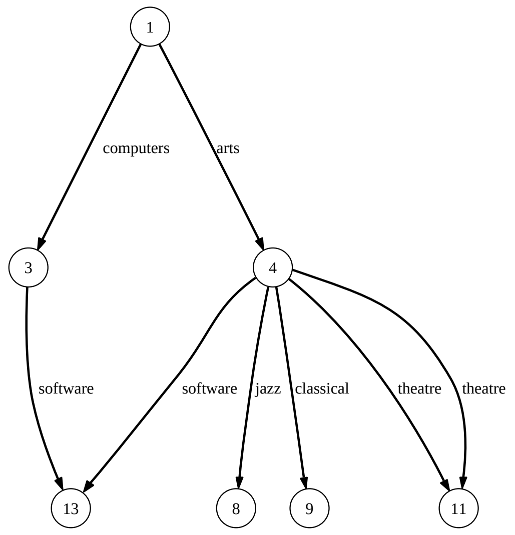


Figure 4: (left) Result of applying select paths to the DAG $D^{'}$ shown in Fig. 2 (right) wrt the term 'music,' i.e., $SP(D^{'},\ \text{music})$. (right) Result of interpretation 2 of out-of-turn interaction with the DAG D shown in Fig. 1 wrt the term 'music,' i.e., $OOT_2(D,\ \text{music})$. Alternatively, one can think of this DAG as the result of shrink edges with the DAG $D^{'''}$ (left), i.e., $SE(D^{'''},\ \text{music})$.

$$OOT_1(D,\ t) = SE(\overbrace{\underbrace{BP(D,\ \overbrace{FP(D,\ t)}^{\text{Fig. 2 (left)}}\,)}_{\text{Fig. 2 (right)}},\ t)}^{\text{Fig. 3}}\,, \tag{1}$$

where SE (*shrink edges*) is a partial function $SE : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{D}_\perp$ which given D and a term $t \in TE(D)$ returns $D^{'}$, where any edge e in D labeled with t is removed in $D^{'}$, the source v_s of e is replaced with its target v_t in $D^{'}$, and v_t becomes the new target of any edge $e^{'}$ with target v_s in $D^{'}$.

Interpretation 2 of out-of-turn interaction is a partial function $OOT_2 : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{D}_\perp$ which given D and a term $t \in TE(D)$ returns $D^{'}$. It is defined as

$$OOT_2(D,\ t) = \underbrace{SE(\overbrace{SP(BP(D,\ FP(D,\ t)),\ t)}^{\text{Fig. 4 (left)}})}_{\text{Fig. 4 (right)}}\,, \tag{2}$$

where SP (*select paths*) is a partial function $SP : (\mathcal{D} \times T) \rightarrow \mathcal{D}_\perp$ which given D and a term $t \in TE(D)$ returns $D^{'}$, where $D^{'}$ contains only the browsing interaction sequences from D involving t (i.e., $SQ(SP(D,t)) = GS(t,\ SQ(D)))$. All browsing interaction sequences pruned under OOT_1 also are pruned under OOT_2, but the converse does not hold. Formally, $SQ(OOT_2(D,\ t)) \subseteq SQ(OOT_1(D,\ t))$. Interpretation 2 retains shrunken versions of only browsing interaction sequences involving the out-of-turn input, while interpretation 1 retains shrunken versions of all browsing interaction sequences which classify the leaf vertices classified by sequences involving the out-of-turn input.

The result of FP is the set of all leaf vertices classified by the browsing interaction sequences involving the out-of-turn input. We back-propagate from this set of leaves up to the root of the DAG with BP. To generalize this approach, we can replace FP with SL (*select leaves*) – any total function $SL : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{L}$ which given D and a term from D returns a set of leaf vertices of D. FP is an instance of SL. The use of a function such as SL allows us to marry our approach with standard techniques from information retrieval (IR) (Baeza-Yates and Ribeiro-Neto, 1999). Moreover, our inclusion of a function which returns a set of leaf vertices leads to the possibility of bringing units of terminal information (additional terms modeled in the leaf documents and not explicitly used in the classification), in replacement of or in addition to structural information, to bear upon the interaction. For instance, we might perform a query (e.g., 'Picasso') in a vector-space model over the set of leaf webpages (documents) using cosine similarity to arrive at a target set of leaves from which to back-propagate. We do not study this extension here and present SL primarily as a technique which works with structural information.

Generalized interpretation 1 of out-of-turn interaction is a partial function $GOOT_1 : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{D}_{\perp}$ which given D and a term $t \in TE(D)$ returns D'. It is defined as

$$GOOT_1(D,\ t) = SE(BP(D,\ SL(D,\ t)),\ t). \tag{3}$$

Generalized interpretation 2 of out-of-turn interaction is a partial function $GOOT_2 : (\mathcal{D} \times \mathcal{T}) \rightarrow \mathcal{D}_{\perp}$ which given D and a term $t \in TE(D)$ returns D'. It is defined as

$$GOOT_2(D,\ t) = SE(SP(BP(D,\ SL(D,\ t)),\ t),\ t) \tag{4}$$

Notice that equations 3 and 4 are the analogs of equations 1 and 2, respectively, in that FP in the latter is replaced by SL. Note also that the out-of-turn paradigm studied here is supported by all four interpretations of out-of-turn interaction.
Notice also that SE can be made optional in each interpretation of out-of-turn interaction. The absence of SE however requires the user to ultimately browse to reach leaf content pages. This idea is generalized below through a concept we call *web functional dependencies.*

Lemma 2.1. *Any interpretation of out-of-turn interaction is commutative, assuming both sides are defined, i.e.,*

$$OOT_1(OOT_1(D,\ x),\ y) = OOT_1(OOT_1(D,\ y),\ x),$$

$$OOT_2(OOT_2(D,\ x),\ y) = OOT_2(OOT_2(D,\ y),\ x),$$

$$GOOT_1(GOOT_1(D,\ x),\ y) = GOOT_1(GOOT_1(D,\ y),\ x), \text{ and}$$

$$GOOT_2(GOOT_2(D,\ x),\ y) = GOOT_2(GOOT_2(D,\ y),\ x),$$

where x and y represent terms. A sketch of the proof of Lemma 2.1 is given in (Perugini, 2004, Ch. 4). While we do not study this extension here, Lemma 2.1 is important to supporting multiple terms per utterance, i.e., the ability to communicate more than one term to the system in a single interaction, an important ingredient for personalized interaction.

3. Program-theoretic View of Personalized Interaction

We now view the above graph-theoretic interpretations of out-of-turn interaction through a programmatic lens and, specifically, relate out-of-turn interaction, and generalizations thereof, to *program slicing*. We begin by illustrating how to model interaction with a hierarchical website as a program.

3.1. Modeling Interaction Programmatically

Researchers have predominately modeled web interaction programmatically to maintain state across and within sessions (Graunke et al., 2001; Queinnec, 2000). Here we model interaction programmatically to personalize information access. Table 4 (left) illustrates a programmatic model of a user's browsing interactions with Fig. 1. Notice our use of procedures to model common sub-trees induced by symbolic links. A *symbolic link* is a hyperlink to a webpage which has an existing incoming hyperlink (Perugini, 2008). For example, the edge from vertex 3 to 4 in Fig. 1 labeled 'speakers' is a symbolic link. The absence of symbolic links from a website make its DAG model a tree. In voluminous web directories, such as *Yahoo!* and ODP, symbolic link labels are augmented with '@' and transport a user from one sub-branch of the directory (e.g., News) to another (e.g., Sports).

The expressive constructs, most notably, nested conditionals, of programming languages make such languages an attractive vocabulary of discourse for modeling interaction with hierarchical hypermedia. While we can model interaction with (imperative, functional, or logic) programming languages, we use C here for purposes of presentation. At the NATO Conference on Software Engineering Techniques held in Rome, Italy in 1969, A. Perlis said, 'a good programming language is a conceptual universe for thinking about programming' (Randell and Buxton, 1969). Similarly here, a good programming language is a conceptual universe for thinking about interacting with hierarchical hypermedia.

Table 4: Modeling interaction programmatically. (left) $\mathtt{P_D}$, programmatic representation of interaction with the website modeled by the DAG D in Fig. 1. (center) $\mathtt{P_{D'}}$, result of applying the zoom transformation to the program on the left wrt `music`. This program is the representation of interaction with the website modeled by the DAG $D^{'}$ in Fig. 2 (right). (right) $\mathtt{P_{D''}}$, result of applying partial evaluation to (center) wrt $\mathtt{music} = 1$. This program is the representation of interaction with the website modeled by the $D^{''}$ in Fig. 3.

$\mathtt{P_D}$	$\mathtt{P_{D'}} = [\![\mathtt{zoom}]\!][\mathtt{P_D},\ \mathtt{music}]$	$\mathtt{P_{D''}} = [\![\mathtt{mix}]\!][\mathtt{P_{D'}},\ \mathtt{music} = 1]$
`if (arts)`	`if (arts)`	`if (arts)`
`  if (theatre)`	`  if (theatre)`	`  if (theatre)`
`    if (drama)`		
`      page = 10;`		
`    if (music)`	`    if (music)`	
`      f1();`	`      f1();`	`      f1();`
`  if (music)`	`  if (music)`	
`    f2();`	`    f2();`	`    f2();`
`if (computers)`	`if (computers)`	`if (computers)`
`  if (speakers)`	`  if (speakers)`	`  if (speakers)`
`    f2();`	`    f2();`	`    f2();`
`  if (software)`	`  if (software)`	`  if (software)`
`    if (music)`	`    if (music)`	
`      f3();`	`      f3();`	`      f3();`
`    if (business)`		
`      page = 14;`		
`  if (hardware)`		
`    if (memory)`		
`      page = 12;`		
`void f1() {`	`void f1() {`	`void f1() {`
`  page = 11;`	`  page = 11;`	`  page = 11;`
`}`	`}`	`}`
`void f2() {`	`void f2() {`	`void f2() {`
`  if (theatre)`	`  if (theatre)`	`  if (theatre)`
`    f1();`	`    f1();`	`    f1();`
`  if (classical)`	`  if (classical)`	`  if (classical)`
`    page = 9;`	`    page = 9;`	`    page = 9;`
`  if (jazz)`	`  if (jazz)`	`  if (jazz)`
`    page = 8;`	`    page = 8;`	`    page = 8;`
`  if (software)`	`  if (software)`	`  if (software)`
`    f3();`	`    f3();`	`    f3();`
`}`	`}`	`}`
`void f3() {`	`void f3() {`	`void f3() {`
`  page = 13;`	`  page = 13;`	`  page = 13;`
`}`	`}`	`}`

Table 5: Illustration of program slicing (simplified for purposes of presentation). (left) A program which accepts the radius and height of a cylinder as input and computes and prints the cylinder's surface area and volume. (center) A static backward slice wrt (6, `vol`). (right) A static forward slice wrt (1, `h`) (variable key: `r` = radius; `h` = height; `cArea` = circle area; `sArea` = surface area).

original program		backward(6,`volume`)	forward(1,`h`)
1	`read(r,h);`	`read(r,h);`	`read(r,h);`
2	`cArea = PI*r*r;`	`cArea = PI*r*r;`	
3	`sArea = 2*cArea+2*r*PI*h;`		`sArea = 2*cArea+2*r*PI*h;`
4	`volume = cArea*h;`	`volume = cArea*h;`	`volume = cArea*h;`
5	`print(sArea);`		`print(sArea);`
6	`print(volume);`	`print(volume);`	`print(volume);`

3.2. Program Slicing

We relate the four interpretations of out-of-turn interaction to the application of program slicing, a transformation common in debuggers, to suitably selected programmatic representations of interaction, such as that shown Table 4 (left). Program slicing (Binkley and Gallagher, 1996; Harman and Hierons, 2001; Tip, 1995), originally introduced by Weiser (1979, 1982, 1984), is a technique used to extract statements, which may affect or be affected by the values of variables of interest computed at some point of interest, from a program. A slice of a program is taken wrt a (point of interest, variable of interest) pair, referred to as the *slicing criterion*. The point of interest may be specified with a line number from the program. The resulting slice consists of all program statements which may affect or be affected by the value of the variable at the specified point.

Slices such as that shown in Table 5 (center), which Weiser first articulated, are called *executable backward static slices* (Binkley and Gallagher, 1996; Horwitz et al., 1990; Venkatesh, 1991) (referred to here as simply *backward slices*). They are *executable* because the slice is required to be an executable program. The slice is *backward* since this is the direction in which dependencies are followed to sources in the program. Contrast this with a *forward slice* (Horwitz et al., 1990) which consists of the program statements affected by the value of a particular variable at a particular statement (see Table 5, right). Backward slices contain data and control predecessors, while forward slices consist of data and control successors. The slice is *static* because it is computed without consideration of program input. Dynamic slicing techniques are covered in (Perugini, 2004, appendix B). For an introduction to program slicing, techniques for computing slices, and applications, we refer the reader to (Binkley and Gallagher, 1996; Harman and Hierons, 2001; Tip, 1995).

We relate FP to the program intersection of two unions of forward slices. Researchers in the programming languages community consider a union of slices as a slice and trivial to compute. The slicing criteria 'can be easily extended to slicing wrt a collection of locations and a collection of variables at each location by taking the union of the individual slices' (Binkley and Gallagher, 1996). The first union consists of the forward slices of the program wrt the variable modeling the out-of-turn input at every occurrence of it in the program. The second union consists of the forward slices of the program wrt the variable indexing the leaf vertices at every occurrence of it in the program. The intersection of these two unions results in several occurrences of the variable indexing the leaf vertices (e.g., `page` in Table 4). Each occurrence is at a point in the program which is affected by the variable corresponding to the out-of-turn input. In graph terms, this procedure results in the set of leaf vertices classified by all browsing interaction sequences involving the out-of-turn input.

The set of program fragments, thus obtained from FP, can be thought of as slicing criteria input to BP which is then the union of backward slices, each wrt every (point, variable) pair resulting from the initial forward slicing procedure. Intuitively, given valid input, a forward slice is performed wrt the corresponding program variable to determine the terminal webpages which are reachable from that point. These webpages are collected and back-propagated with backward slicing so that only those paths that reach these pages are retained. Notice that these two operations implicitly capture exclusions among program variables, e.g., when the user says 'Honda' the slices remove any program segments which involve Toyotas. Such a combination of forward and backward slicing is similar to the zoom operator for interactively pruning information hierarchies (Sacco, 2000). Thus, we call this composite program transformation technique *zoom*.

The idea of performing set-theoretic operations on forward and backward slices is closely related to the concepts of *program dicing* (Binkley and Gallagher, 1996; Lyle and Weiser, 1987) and *program chopping* (Jackson and Rollins, 1994b). Performing set-theoretic operations on one or more backward program slices yields a *program dice* (Binkley and Gallagher, 1996; Lyle

Table 6: Comparison of partial evaluation and program slicing along a syntax- vs. semantic-preserving dichotomy.

	Syntax-preserving	**Semantic-preserving**
Partial evaluation	$\times$	$\surd$
Program slicing	$\surd$	$\times$

and Weiser, 1987). Originally program dicing was limited to backward slices. Program chopping, on the other hand, which also is a generalization of slicing (Jackson and Rollins, 1994b), is an extension of dicing to forward slices. Forward slices increase the usefulness of dicing (Binkley and Gallagher, 1996). Chopping identifies the statements which transmit values from one statement to another. In other words, it shows all the ways which one set of program points affect another set of points. A *program chop* (Jackson and Rollins, 1994b,a; Reps and Rosay, 1995) therefore consists of all program points affected between one point (the chop source) and another (the chop target) (Anderson et al., 2003). It also is the subset of the intersection of a forward and backward slice (Anderson, P., personal communication, November 10, 2003). In the absence of procedures, a chop 'can be viewed as a generalized kind of program dice' (Binkley and Gallagher, 1996).

Table 6 compares partial evaluation and program slicing. It reinforces that while partial evaluation is semantic-preserving, it is not syntax-preserving. Conversely, while the variants of program slicing considered in this article are syntax-preserving, they are not semantic-preserving. However, there are variants of program slicing (e.g., *amorphous slicing*, also known as *semantic slicing*) which are the reverse, i.e., semantic-preserving, but not syntax-preserving (see Perugini, 2004, appendix B for more details).

3.3. Notation: Programs as Data Objects

To succinctly describe our program transformation techniques (e.g., the above notion of zoom) we adopt a slightly modified notation for describing the semantic function of a programming language used in a textbook on partial evaluation (Jones et al., 1993). A specification language, defined by a context-free grammar, for program transformations is introduced in (Hildum and Cohen, 1990). While the language is imperative, here, for purposes of presentation, we use a declarative style. GrammaTech, Inc., the company which develops and produces a program slicer for ANSI C called *CodeSurfer* is currently working on a textual representation, employing a Lisp-like syntax, for set-theoretic operations over program slices for a future release, e.g., (intersect (slice A) (slice B)) (Anderson, P., personal communication, April 8, 2004).

- $[\![\mathtt{int}]\!][\mathtt{P_D}, \mathtt{x_1} = \mathtt{1}, \mathtt{x_2} = \mathtt{0}, \ldots, \mathtt{x_n} = \mathtt{1}]$ denotes 'partially, interpret the programmatic representation of DAG D ($\mathtt{P_D}$) wrt the partial assignment of variables $\mathtt{x_1} = \mathtt{1}$, $\mathtt{x_2} = \mathtt{0}$, $\ldots$, $\mathtt{x_n} = \mathtt{1}$.' $[\![\mathtt{int}]\!]$ denotes the application of a stepwise interpreter. We use interpretation here as a program transformation.

- $[\![\mathtt{mix}]\!][\mathtt{P_D}, \mathtt{x_1} = \mathtt{1}, \mathtt{x_2} = \mathtt{0}, \ldots, \mathtt{x_n} = \mathtt{1}]$ denotes 'partially evaluate (non-sequentially interpret) the programmatic representation of DAG D ($\mathtt{P_D}$) wrt the partial assignment of variables $\mathtt{x_1} = \mathtt{1}$, $\mathtt{x_2} = \mathtt{0}$, $\ldots$, $\mathtt{x_n} = \mathtt{1}$.' $[\![\mathtt{mix}]\!]$ is the conventional way to denote a partial evaluator (Jones, 1996, 1997; Jones et al., 1993). It refers to *mixed computation* since a partial evaluator performs a mixture of interpretation and code-generation (Jones, 1996).

- $[\![\mathtt{forward}]\!][\mathtt{P_D}, \mathtt{x}]$ denotes 'union each forward slice of the programmatic representation of DAG D ($\mathtt{P_D}$) wrt the variable $\mathtt{x}$ at every program point containing $\mathtt{x}$.'

- $[\![\mathtt{backward}]\!][\mathtt{P_D}, \mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}]$ denotes 'union each backward slice of the the programmatic representation of DAG D ($\mathtt{P_D}$) wrt the variable $\mathtt{x}$ at program points $\mathtt{1}, \mathtt{2}, \ldots, \mathtt{n}$.'

Using this notation, $[\![zoom]\!]$ is formally defined as

$$[\![\mathtt{zoom}]\!][\mathtt{P_D},\ \mathtt{input}] = [\![\mathtt{backward}]\!][\mathtt{P_D},\ [\![\mathtt{forward}]\!][\mathtt{P_D},\ \mathtt{input}] \cap [\![\mathtt{forward}]\!][\mathtt{P_D},\ \mathtt{page}]],$$

where `input` is the program variable representing the out-of-turn input and `page` is the variable indexing the leaf vertices. We define $[\![sp]\!]$, the programmatic analog to *select paths* (SP), as

$$[\![\mathtt{sp}]\!][\mathtt{P_D},\ \mathtt{x}] = [\![\mathtt{forward}]\!][\mathtt{P_D},\ \mathtt{x}] \cup [\![\mathtt{backward}]\!][\mathtt{P_D},\ \mathtt{x}].$$

We define $[\![\mathtt{te}]\!][\mathtt{P_D}]$, the programmatic analog to *term extraction* (TE), as the union of all program data successors of each structural variable at its declaration. A program data successor is a restriction to a forward slice in that rather than including transitive dependencies, it just contains the

Table 7: Relating interaction techniques in DAG models of a websites to compositions of program transformations. Notice that $[\![\mathrm{SL}]\!]$ is a *meta*-program-transformation, i.e., it represents any program transformation which returns a set of program points containing the variable `page`.

Interaction technique	**Program transformation technique**
Browsing	$[\![\texttt{int}]\!]$ $[\mathrm{P_D}$, `input = 1`$]$
Interp. 1 of OOT Interaction	$[\![\texttt{mix}]\!]$ $[[\![\texttt{zoom}]\!]$ $[\mathrm{P_D}$, `input`$]$, `input = 1`$]$
Interp. 2 of OOT Interaction	$[\![\texttt{mix}]\!]$ $[[\![\texttt{sp}]\!]$ $[\mathrm{P_D}$, `input`$]$, `input = 1`$]$
Gen. Interp. 1 of OOT Interaction	$[\![\texttt{mix}]\!]$ $[[\![\texttt{backward}]\!]$ $[\mathrm{P_D}$, $[\![\mathrm{SL}]\!]$ $[\mathrm{P_D}$, `input`$]]$, `input = 1`$]$
Gen. Interp. 2 of OOT Interaction	$[\![\texttt{mix}]\!]$ $[[\![\texttt{sp}]\!]$ $[[\![\texttt{backward}]\!]$ $[\mathrm{P_D}$, $[\![\mathrm{SL}]\!]$ $[\mathrm{P_D}$, `input`$]]$, `input`$]$, `input = 1`$]$

immediate dependency of a program point. Specifically, 'a program point's data successors are the points where the variables that were modified at that point are used' (Anderson et al., 2003). Since structural variables' sole presence in these programs arises in the context of an `if (...)` expression, we might just as easily think of conducting $[\![\texttt{te}]\!][\mathrm{P_D}]$ with a regular expression.

Armed with these formalisms, we can relate interpretations of interaction techniques, including browsing, to classical program transformations. In Table 7 we present program transformation techniques to realize browsing and the four interpretations of out-of-turn interaction given above. Program slicing is typically used for debugging, safety, and security. Only few have used slicing for web applications (Ricca and Tonella, 2001b). Our use of program slicing helps marry it with information personalization.

4. Evaluation

A *model* $\mathcal{M} = (P_D,\ \mathcal{X})$ is a program (representation, transformation) pair. We would like to evaluate a model by assessing its capability to realize a desired set of interaction sequences. To do so we measure how close the model comes to realizing the targeted interaction paradigm $\mathcal{P}$. Ideally, we would like to have

$$\mathcal{M} \rightsquigarrow I \leftrightarrow I \in \mathcal{P},$$

where $\rightsquigarrow$ denotes *stages*, i.e., $\mathcal{M}$ stages interaction sequence I *iff* I is in the interaction paradigm $\mathcal{P}$. A model stages an interaction sequence if successive applications of its transformation technique to its programmatic representation, given user input, realize the interaction sequence. In this manner, the model *stages* the user's interaction.

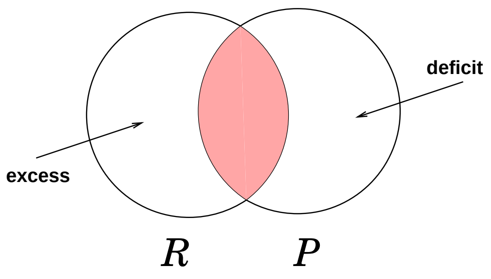


Figure 5: Venn diagram highlighting the intersection between the set of sequences R (left) staged by an incomplete, unsound model and its intended interaction paradigm $\mathcal{P}$ (right).

Soundness of a Model: $\mathcal{M} \rightsquigarrow I \rightarrow I \in \mathcal{P}$

A model $\mathcal{M}$ is *sound* for an interaction paradigm $\mathcal{P}$ if each interaction sequence which $\mathcal{M}$ stages is in $\mathcal{P}$. In other words, if the model can stage an interaction sequence, then the sequence is in the paradigm.

Completeness of a Model: $I \in \mathcal{P} \rightarrow \ \mathcal{M} \rightsquigarrow I$

A model $\mathcal{M}$ is *complete* for an interaction paradigm $\mathcal{P}$ if $\mathcal{M}$ can stage each interaction sequence in $\mathcal{P}$. In other words, if an interaction sequence is in the paradigm, then the model can stage it.

The models presented in Table 7 are each sound and complete for the corresponding out-of-turn paradigm presented in this article. The completeness of each model holds under the assumption that no path from the root of the website to each leaf contains more than one hyperlink with the same label. Intuitively, this is because communicating (in-turn or out-of-turn) information initiates a transformation which simplifies the site wrt all hyperlinks labeled with that input, some of which may lie along the same path. This assumption is captured by our definition of interaction set and (browsing) interaction sequence. As a result, our definition of browsing is slightly different than its traditional interpretation. The soundness of the model holds under the assumption that all terms supplied in any interaction sequence lie along a single path. This is always true in a tree, but may not be true in a DAG. This assumption is captured by the definition of interaction set and (out-of-turn) interaction sequence.

We can identify both the excess and deficit of a model. A complete, but unsound, model has *excess* – interaction sequences not in its intended paradigm which it stages. On the other hand, a sound, but incomplete, model has *deficit* – sequences in its intended interaction paradigm which it fails to stage. An unsound and incomplete model exhibits both excess and deficit wrt its targeted paradigm. A *sufficiency* metric for a model is formulated akin to the recall measure in IR (Baeza-Yates and Ribeiro-Neto, 1999):

$$\mathit{sufficiency} = \frac{|R \cap \mathcal{P}|}{|\mathcal{P}|} = \frac{|R \cap \mathcal{P}|}{SOP(D)},$$

where R represents the set of sequences staged by the model and

$$SOP(D) = \sum_{I_i \in SQ(D)} |gIS(I_i)|!,$$

where gIS (*get Interaction Set*) is a total function $gIS : \mathcal{I} \rightarrow \mathcal{S}$ which given an interaction sequence I_i returns the interaction set over which it is defined. We use $\mathcal{S}$ to denote the universal set of sets. Fig. 5 illustrates how excess and deficit arise.

Lemma 4.1. *Any complete model for an out-of-turn interaction paradigm is also complete for a browsing paradigm.*

A sketch of the proof of Lemma 4.1 is given in (Perugini, 2004, Ch. 4). Intuitively, this lemma indicates any browsing paradigm is a proper subset of the corresponding out-of-turn interaction paradigm or, in other words, out-of-turn interaction subsumes browsing. *This is a significant result as it means that we can support the union of these two interaction paradigms with a single program transformation technique. In other words, no anticipation of in-turn or out-of-turn input is necessary to discern a program transformation technique. Thus, we achieve uniform processing of input.* For interaction this means that the user can interleave hyperlink clicks (in-turn browsing inputs) and voice utterances (out-of-turn inputs) in any order she desires (Perugini and Ramakrishnan, 2003b) to achieve a *mixed-initiative* mode of information seeking (Allen, 1999). *Mixed-initiative interaction* is a flexible interaction strategy where each participant can play an equal role in steering the progression of the interaction (Hearst, 1999).

Beyond soundness and completeness, we developed a measure which estimates the *compression* achieved in a model – the ratio of interaction sequences realizable through interpretation (and hence directly stated in the representation) to the total number of sequences realizable through transformation:

$$compression = \frac{|R - E|}{|R|},$$

where E represents the set of interaction sequences stageable from P_D with an interpreter ($[\![\texttt{int}]\!]$). Intuitively, the compression ratio quantifies the percentage of sequences which we get 'for free' by using the program transformation technique.

An effective model is one which maximizes both sufficiency and compression. However, these measures are bipolar and exhibit a tradeoff similar to that between *precision* and *recall* in IR (Baeza-Yates and Ribeiro-Neto, 1999). To maximize sufficiency, we might choose to explicitly model each interaction sequence in the representation, affecting the compression ratio negatively.

There are alternate applicable evaluation criteria for this research (Ramakrishnan and Perugini, 2001). Evaluating personalization applications for traditional user-satisfaction and task completion metrics is another practice. In addition to measuring satisfaction, studies with users can improve our understanding of an extant interaction paradigm or yield new paradigms to model (Perugini, 2004, Ch. 6) (Perugini et al., 2007). Personalization applications also can be evaluated wrt classical IR metrics (Dhyani et al., 2002). For example, Sacco (2000) studies how the application of zoom as well as bucket size (i.e., number of documents classified under each terminal concept) affect the maximum resolution of a taxonomy. Maximum resolution, which is a measure of retrieval effectiveness, is the average minimum number of documents the user has to manually inspect.

5. Graph-theoretic Classes of Hierarchical Hypermedia

While the transformation techniques in Table 7 work on any DAG, we identify classes of hierarchical hypermedia for insight into the possibility of using specialized program transformation techniques to realize out-of-turn interaction for each class. We begin by defining a few terms.

The *maximum depth* of a DAG D is the level of an edge-label in D which is greater than or equal to the level of all other edge-labels in D. A *cluster* c is a term-level set such that no edge label in it labels an edge in a different term-level set. If the maximum depth of D equals the number of clusters in D, then D is *levelwise* DAG. Intuitively, a levelwise DAG is one where

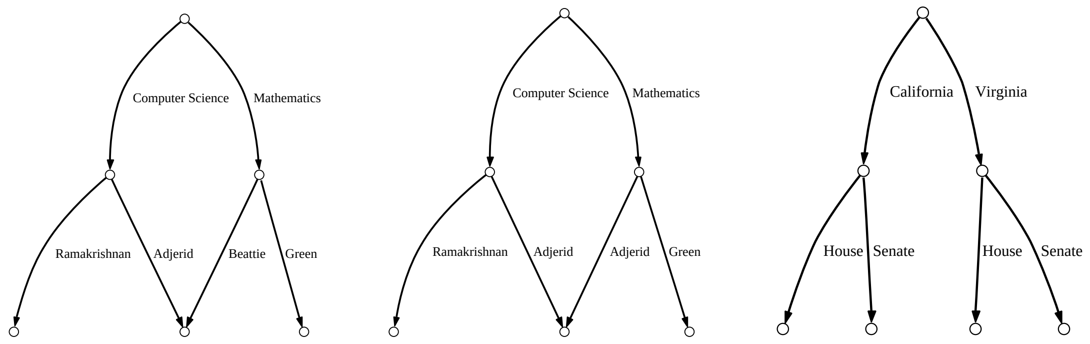


Figure 6: Simple levelwise DAGs. (left) non-mutually exclusive. (center) weak-mutually exclusive. (right) strong-mutually exclusive.

each level of the DAG corresponds to a *facet* (Taylor, 2000) of information assessment in the website it models or, in other words, each term $t_i \in TE(D)$ resides at exactly one level. Therefore, sometimes such sites are referred to as *faceted*.

Each DAG shown in Fig. 6 is levelwise. The DAG in Fig. 6 (left) models a simplified path through the online Virginia Tech timetable of courses. Its clusters are {Computer Science, Mathematics} and {Ramakrishnan, Adjerid, Beattie, Green} implying that the two levels shown correspond to *department* and *instructor* facets, respectively. Notice that symbolic links, such as those illustrated in Fig. 6 (left and center) by the edge labeled 'Adjerid' emanating from the target of the edge labeled 'Computer Science,' are necessary to model the presence of cross-listed courses. The DAG in Fig. 6 (right), which models the Congressional portion of the Project Vote Smart (PVS at `http://votesmart.org`) website, also is levelwise, albeit without symbolic links. Its clusters are {California, Virginia} and {House, Senate} implying that the two levels shown correspond to *state* and *branch of Congress* facets, respectively. In contrast to those shown in Fig. 6, notice that the DAG in Fig. 1 is non-levelwise, or sometimes called *unfaceted* (Perugini, 2009).

We now introduce the concept of mutual exclusivity in DAG models of websites. If no leaf vertex of D lies at the end of two paths from D which each involve a distinct edge-label from a term-co-occurrence set T, then we say that T is a *mutually exclusive* term-co-occurrence set. While there are multiple mutually exclusive term-co-occurrence sets (e.g., {Computer Science, Green}) in the DAGs shown in Fig. 6 (left and center), none are clusters.

On the other hand, the term-level sets {California, Virginia} and {House, Senate} of the DAG shown in Fig. 6 (right) are mutually exclusive and clusters. If D is levelwise and has at least one mutually exclusive cluster, then D is a *weak-mutually exclusive* DAG. If a DAG D is levelwise and no leaf vertex of D lies at the end of two distinct paths from D, where each path contains a distinct term from the same cluster, then D is *strong-mutually exclusive* DAG. Note that the mutually exclusivity of DAGs subsumes the levelwise property by definition. Notice further that replacing only one edge-label (i.e., 'Beattie' to 'Adjerid') in the DAG shown in Fig. 6 (left) makes it a weak-mutually exclusive DAG (see Fig. 6, center); its cluster at level-two is mutually exclusive, but that at level one is not due to the symbolic link. This DAG models a course in the online Virginia Tech timetable which presumably has two sections, each taught by a different instructor, from different departments. If there is a unique simple path from the root of a DAG D to each vertex in D, then D is an *edge-labeled, rooted tree* (hereafter referred to as a *tree*), e.g., Fig. 6 (right). Observe also that we develop the above classes of hierarchical hypermedia without reference to semantics; these notions are purely syntactic, e.g., the levelwise property does not take into account polysemy of terms.

Lemma 5.1. *A DAG D is levelwise and each term-level set of D is a cluster and mutually exclusive iff D is strong-mutually exclusive.*

Lemma 5.2. *A strong-mutually exclusive DAG which is not a tree does not exist, given our definition of a DAG.*

Lemma 5.3. *Interpretation 1 of out-of-turn interaction over a tree D is functionally equivalent to interpretation 2 of out-of-turn interaction over D.*

Lemma 5.4. *Generalized interpretation 1 of out-of-turn interaction over a tree D is functionally equivalent to generalized interpretation 2 of out-of-turn interaction over D.*

Lemma 5.5. *The interpretations of out-of-turn interaction considered in this article preserve the levelwise property in DAGs.*

Lemma 5.6. *If a DAG D is a levelwise tree, then D is a strong-mutually exclusive tree.*

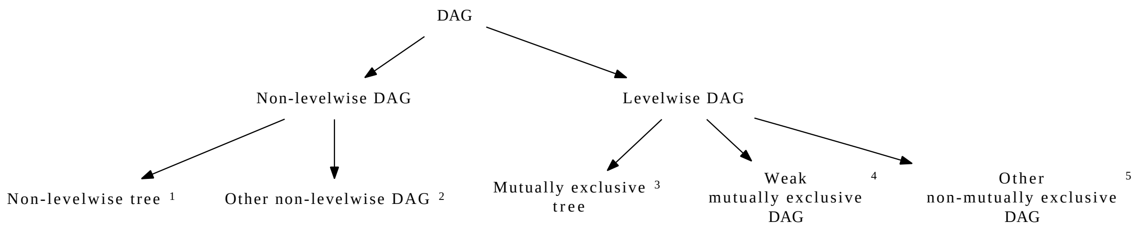

DAG
Non-levelwise DAG
Levelwise DAG
Non-levelwise tree 1
Other non-levelwise DAG 2
Mutually exclusive tree 3
Weak mutually exclusive DAG 4
Other non-mutually exclusive DAG 5

Figure 7: Partial order of classes of hierarchical hypermedia. Superscripts help connect these classes to those in Table 8.

Table 8: Alternate view of classes of hierarchical hypermedia making the five considered classes (leaves of the tree in Fig. 7) salient. Superscripts help connect these classes to those in Fig. 7. Legend: lw = levelwise, me = mutually exclusive; symbols $\surd$ and $\perp$ denote defined and undefined, respectively.

DAG	**non-me**	**me**
non-lw	$\surd^2$	$\perp$
lw	$\surd^5$	$\surd^4$

Tree	**non-me**	**me**
non-lw	$\surd^1$	$\perp$
lw	$\perp$	$\surd^3$

Corollary 1. *The interpretations of out-of-turn interaction considered in this article preserve the strong-mutually exclusive property in trees.*

Lemmas 5.2 and 5.6 make the type of mutually exclusivity unequivocal in DAGs. Thus, unless stated otherwise, we refrain from qualifying mutually exclusivity as *weak* or *strong* in such contexts.

Sketches of the proofs of Lemmas 5.1–5.6 are given in (Perugini, 2004, Ch. 4). The proof for 5.4 follows analogously from Lemma 5.3. We can construct its outline by replacing all occurrences of FP in the proof of Lemma 5.3 with SL, since FP is an instance of SL. The proof of Corollary 1 follows directly from Lemmas 5.5 and 5.6.

A Hasse diagram for a partial order of these classes of hierarchical hypermedia is shown in Fig. 7. The leaf vertices in this Hasse diagram are the five classes considered here. Examples of sites modeled after non-levelwise trees are GAMS at `http://gams.nist.gov` and the *Google* directory at `http://directory.google.com`. Examples of other non-levelwise DAGs are the *Yahoo!* directory and ODP.[2] PVS and the Kelley Blue Book at `http://kbb.com`

[2]Formally, the *Yahoo!* directory and ODP are not DAGs due to the presence of cycles

each can be modeled as a mutually exclusive tree. The Virginia Tech online timetable of classes can be modeled as a weak mutually exclusive DAG. The mutually exclusive tree class is the primary focus of our user studies (Perugini, 2004, Ch. 6) (Perugini et al., 2007). Note that the other non-mutually exclusive DAG class does not contain any trees, by Lemma 5.6, as shown in Fig. 7. An alternate view of these five classes is given in Table 8.

We describe methods for automatically identifying these classes of hierarchical hypermedia in (Perugini, 2004, Ch. 4). The application of any interpretation of out-of-turn interaction to a DAG may result in a mutually exclusive DAG. Therefore, to take advantage of program transformation techniques specialized for these classes, which we develop below, we need to apply these identification procedures after each out-of-turn interaction.

6. Mining Web Functional Dependencies for Automatic Query Expansion

Before re-entering the program-theoretic domain, we must develop a few more graph-theoretic terms and tools which help provide intuition for the relations to some of the program transformation techniques to follow. Specifically, we illustrate how mining dependencies underlying websites helps expand input for out-of-turn interaction.

A *path-term-co-occurrence set* is a term-co-occurrence set which only contains the terms from D which co-occur with a particular term along paths through D. *Build path-term-co-occurrence set* is total function $BPTC$: $(\mathcal{D} \times T) \rightarrow P(T)$ which given D and a term $t \in T = TE(D)$ returns a path-term-co-occurrence set. It is defined as

$$BPTC(D,\ t) = \bigcup_{I_{t_i}\ \in\ GS(t,\ SQ(D))} GIS(I_{t_i}) - \{t\},$$

where I_{t_i} is an interaction sequence from D containing term t_i. The set {Adjerid, Green} is the path-term-co-occurrence set in the DAG illustrated in Fig. 6 (center) wrt the term 'Mathematics.'

Given a set of leaves from D reachable through paths containing a particular term t (i.e., $L_t = FP(D,\ t)$), a *leaf-term-co-occurrence set* is a term-co-occurrence set which only contains the terms from D, excluding t, which

induced by symbolic links.

lie along all of the paths from D which reach all of the leaves in L_t. *Build leaf-term-co-occurrence set* is total function $BLTC : (\mathcal{D} \times T) \rightarrow P(T)$ which given D and term $t \in T = TE(D)$ returns a leaf-term-co-occurrence set. It is defined as

$$BLTC(D,\ t) = TE(BP(FP(D,\ t))) - \{t\}.$$

The set {Computer Science, Adjerid, Green} is the leaf-term-co-occurrence set in the DAG illustrated in Fig. 6 (center) wrt the term 'Mathematics.'

Lemma 6.1. *If a DAG D is a tree, then any path-term-co-occurrence set wrt a term t in D is a leaf-term-co-occurrence set wrt t in D. In other words, if a DAG D is a tree, then $BPTC(D,\ t) = BLTC(D,\ t)$.*

A sketch of the proof of Lemma 6.1 is given in (Perugini, 2004, Ch. 4).

Corollary 2. *If a DAG D is a tree, then the complete set of path-term-co-occurrence sets in D equals the complete set of leaf-term-co-occurrence sets in D. In other words, if a DAG D is a tree, then*

$$\bigcup_{t_i\ \in\ TE(D)} \{BPTC(D,\ t_i)\} = \bigcup_{t_i\ \in\ TE(D)} \{BLTC(D,\ t_i)\}.$$

The proof for Corollary 2 follows directly from Lemma 6.1.

Identifying structural relationships in data-intensive websites, in domains such as e-commerce (e.g., Amazon), digital libraries (e.g., CITIDEL at `http://citidel.org`), and scientific computing (e.g., GAMS at `http://gams.nist.gov`) is becoming a precursor to personalizing information access (Abiteboul et al., 2000). Deploying out-of-turn interaction involves addressing some practical considerations, including the identification of such relationships. Specifically, when the targeted website contains dependencies between and across the levels of its DAG model, an out-of-turn interaction can result in the system soliciting information from the user which can be inferred from the previous input. For instance, consider a user's interaction with the Kelley Blue Book, an automobile website which progressively solicits for automobile attributes, in an order pre-determined by the site designer, and forces users to communicate those attributes in this manner to access a vehicle webpage. KBB is modeled as mutually exclusive tree. If a user communicates an out-of-turn input which causes all paths save for one to be pruned from the site, then that user is required to click through a series of webpages each only containing one link (leading to the next page). For instance, if saying 'Civic' out-of-turn results in one car, then the user may still have to click the 'sedan' hyperlink on the resulting page and the 'Honda' hyperlink on the penultimate page en route to the leaf webpage.

Classic menu design research states that no menu should contain less than two items. Therefore, in such cases, we would like to consolidate the path and lead the user directly to the terminal webpage in one stroke. Thus, when the user says 'Civic' out-of-turn we can infer 'sedan Honda' by functional dependency and safely expand the input to 'Civic sedan Honda' without losing any information. The 'Civic → Honda' web FD asserts that all of the paths through the site involving 'Civic' also involve 'Honda.' Clearly, the reverse does not hold as Honda makes several automobile models. We call such a dependency a *positive-path web FD*. We detect such dependencies between the site's levels (in this case, vehicle-type, -maker, and -model) and leverage these positive web FDs for query expansion. When such dependencies are employed, it is important to provide real-time feedback to users so that the information contained on the rhs of the web FD is not lost. This can be done by collecting the rhs of each web FD triggered and augmenting the leaf webpage with this information. Alternatively, we could present an 'Input so far:' label in the browser's status bar and, at each step in the interaction, incrementally include the rhs of any fired web FDs. Notice that triggering such web FDs affects the stageable interaction sequences and thus the realizable (browsing and out-of-turn) interaction paradigms. Thus, we may have to update our definition of an interaction sequence so that it can be defined over a subset of an interaction set.

Using web FDs to expand user out-of-turn input ultimately creates invisible shortcuts through the website for the user. For example, saying 'Washington, DC' out-of-turn at the top level of PVS expands to 'Washington, DC House Democrat District-at-large' (because only one congressperson represents Washington, DC) and directly reaches the webpage of an individual congressional official. In levelwise sites, a cursory understanding of the underlying domain is necessary to manually identify a majority of the web FDs. For instance, in KBB, several web FDs fit the $model \rightarrow make$ template (e.g., 'Civic → Honda'). Web FDs are less salient in sites which are non-levelwise such as large web directories such as *Yahoo!* and ODP. Thus, techniques from association rule mining, especially those designed for the web (Eirinaki and Vazirgiannis, 2003; Mobasher et al., 2000; Mulvenna et al., 2000; Srivastava et al., 2000), become important and applicable in both classes.

6.1. Mining Positive Web Functional Dependencies

Positive-path Web Functional Dependencies

Discovering positive-path web FDs entails identifying pairs of hyperlink labels where all of the paths through the site involving one also involve the other. Then when a user supplies out-of-turn input, we consult the precomputed set of web FDs to answer the question: what terms in the complete set of terms, $TE(D)$, lie along each path involving a hyperlink labeled with the out-of-turn input? An instance of such a web FD in KBB is 'Civic $\rightarrow$ Honda.'

Since the number of paths remaining through the site is reduced as a result of each (in-turn or out-of-turn) interaction, the number of positive-path web FDs satisfied by a site dynamically changes after each interaction. Therefore, while the number of such web FDs (say p) in a site is potentially exponential in its size, just as in traditional data mining algorithms, the number of web FDs reduces dramatically as we drill through the levels. We have developed an algorithm, which is output-sensitive with complexity $O(p)$, to mine all of the possible positive-path web FDs a priori (Perugini and Ramakrishnan, 2007).

An alternate approach is to mine a new set of web FDs at each step in the interaction. However, since positive-path web FDs can only exist among terms which co-occur on a path, we need not examine $TE(D)$ terms per term. Rather we can make use of a path-term-co-occurrence set and examine only the terms which co-occur on a path with the particular term (corresponding to the out-of-turn input). This approach implies computing a path-term-co-occurrence set after the user supplies an out-of-turn input and using it to discover any positive-path web FDs relevant to the user's interaction. *Mine positive-path-FD set* is total function $mFDs_{pp} : (\mathcal{D} \times \mathcal{T}) \rightarrow P(T)$ which given D and a term $t_i \in T = TE(D)$ returns a term-co-occurrence set. It is defined as

$$mFDs_{pp}(D,\, t_i) = \bigcup_{t_j \,\in\, BPTC(D,\, t_i)} \{t_j\} \;\; \text{such that } GS(t_i,\; SQ(D)) \subseteq GS(t_j,\; SQ(D)).$$

This function finds web FDs of the form $t_i \rightarrow t_j$. A *positive-path-FD set* of D is any term-co-occurrence set returned from $mFDs_{pp}$. The complete set of positive-path web FDs satisfied by the DAG in Fig. 6 (left) is

{Ramakrishnan	→	Computer Science,
Adjerid	→	Computer Science,
Beattie	→	Mathematics,
Green	→	Mathematics}.

No positive-path web FDs hold in the DAG shown in Fig. 6 (right).

Positive-leaf Web Functional Dependencies

Positive-leaf web FDs are more general than positive-path web FDs in that all positive-path web FDs also are positive-leaf web FDs. Discovering positive-leaf web FDs entails identifying pairs of hyperlink labels where all of the leaves classified under paths involving one label also are classified under paths involving the other. Then when a user supplies out-of-turn input, we consult the set of web FDs to answer the question: what terms in the complete set of terms, $TE(D)$, lie along paths which classify all leaves reachable by paths involving a hyperlink labeled with the out-of-turn input? The number of positive-leaf web FDs in a site changes after every interaction because the number of leaves remaining in the site is reduced as a result of each interaction. Thus, we can mine a new set of positive leaf web FDs at each step. Since positive-leaf web FDs can only exist among terms which co-occur wrt leaves, we need not examine $TE(D)$ terms per term. Rather we can make use of a leaf-term-co-occurrence set and examine only the terms which co-occur with a particular term (corresponding to the out-of-turn input) wrt leaves. This approach implies computing a leaf-term-co-occurrence set after the user supplies an out-of-turn input and using it to discover any positive-leaf web FDs relevant to the user's interaction. *Mine positive-leaf-FD set* is total function $mFDs_{pl} : (\mathcal{D} \times \mathcal{T}) \rightarrow P(T)$ which given D and a term $t_i \in T = TE(D)$ returns a term-co-occurrence set. It is defined as

$$mFDs_{pl}(D,\ t_i) = \bigcup_{t_j\ \in\ BLTC(D,\ t_i)} \{t_j\} \quad \text{such that } FP(D,\ t_i) \subseteq FP(D,\ t_j).$$

This function mines web FDs of the form $t_i \rightarrow t_j$. A *positive-leaf-FD set* of D is any term-co-occurrence set returned from $mFDs_{pl}$. The complete set of positive-leaf web FDs satisfied by the DAG in Fig. 6 (left) is

{Ramakrishnan	→	Computer Science,
Adjerid	→	Computer Science,
Adjerid	→	Mathematics,
Adjerid	→	Beattie,
Beattie	→	Computer Science,
Beattie	→	Mathematics,
Beattie	→	Adjerid,
Green	→	Mathematics}.

The DAG shown in Fig. 6 (right) does not satisfy any positive-leaf web FDs.

Lemma 6.2. *Any positive-path web FD in a DAG D is a positive-leaf web FD in D.*

Corollary 3. *Any positive-path web FD set wrt a term t_i in a DAG D is a subset of the positive-leaf web FD set wrt t_i in D. In other words, $mFDs_{pp}(D,\ t_i) \subseteq mFDs_{pl}(D,\ t_i)$.*

We leave it to the reader to convince thyself that the converse of Lemma 6.2 (i.e., any positive-leaf web FD is a positive-path web FD) is not true. However, we have:

Lemma 6.3. *If a DAG D is a tree, then any positive-leaf web FD in D is a positive-path web FD in D.*

Corollary 4. *If a DAG D is a tree, than any positive-path web FD set wrt a term t_i in D equals the positive-leaf web FD set wrt t_i in D. In other words, if D is a tree, then $mFDs_{pp}(D,\ t_i) = mFDs_{pl}(D,\ t_i)$.*

Corollary 5. *If a DAG D is a tree, then the complete set of positive-path web FDs in D equals the complete set of positive-leaf web FDs in D.*

Notice that Lemma 6.3 uses *if*, not *iff*; we leave it to the reader to convince thyself that the converse of Lemma 6.3 (i.e., if any positive-leaf web FD in a DAG D is a positive-path web FD in D, then DAG D is a tree) is not true.

Identifying and employing instances of positive web FDs is helpful for usability purposes, but not necessary for realizing out-of-turn interaction. What is non-intuitive, however, is that a related concept, *negative web FDs*, is helpful in developing an alternate program transformation technique for out-of-turn interaction and specializations of it for the mutually exclusive classes of hierarchical hypermedia (when we use the phrase *mutually exclusive classes* we refer to a *mutually exclusive tree* and a *weak mutually exclusive DAG*, not a general connotation to two (or more) classes which are mutually exclusive with each other). Specifically, partially evaluating a programmatic representation of a website wrt the variables representing the terms on the rhs of a *negative-path* or *-leaf* FD set each assigned zero prunes the site.

6.2. Mining Negative Web Functional Dependencies

Negative-path Web Functional Dependencies

Identifying negative-path web FDs entails identifying pairs of hyperlink labels where none of the paths through the site involving one label involve the other. Identifying negative-path web FDs involves answering the question: what terms in the complete set of terms, $TE(D)$, *never* lie along any of the paths involving a hyperlink labeled with a particular term? An example of such a web FD in KBB is 'Honda $\rightarrow \neg$ Toyota.' Intuitively, this means that none of the paths through the site involving the term 'Honda' involve the term 'Toyota.' *Mine negative-path-FD set* is total function $mFDs_{np} : (\mathcal{D} \times \mathcal{T}) \rightarrow P(T)$ which given D and a term $t_i \in T = TE(D)$ returns a term-co-occurrence set. It is defined as

$$mFDs_{np}(D,\ t_i) = TE(D) - \{t_i\} - BPTC(D,\ t_i).$$

A *negative-path-FD set* of D is any term-co-occurrence set returned from $mFDs_{np}$. The complete set of negative-path web FDs satisfied by the DAG in Fig. 6 (left) is

{Computer Science	→	¬ Mathematics,
Computer Science	→	¬ Beattie,
Computer Science	→	¬ Green,
Mathematics	→	¬ Computer Science,
Mathematics	→	¬ Ramakrishnan,
Mathematics	→	¬ Adjerid,
Ramakrishnan	→	¬ Mathematics,
Ramakrishnan	→	¬ Adjerid,
Ramakrishnan	→	¬ Beattie,
Ramakrishnan	→	¬ Green,
Adjerid	→	¬ Mathematics,
Adjerid	→	¬ Ramakrishnan,
Adjerid	→	¬ Beattie,
Adjerid	→	¬ Green,
Beattie	→	¬ Computer Science,
Beattie	→	¬ Ramakrishnan,
Beattie	→	¬ Adjerid,
Beattie	→	¬ Green,
Green	→	¬ Computer Science,
Green	→	¬ Ramakrishnan,
Green	→	¬ Adjerid,
Green	→	¬ Beattie}.

Likewise, the complete set of negative-path web FDs satisfied the DAG in Fig. 6 (right) is

{California	→	¬ Virginia,
Virginia	→	¬ California,
House	→	¬ Senate,
Senate	→	¬ House}.

Negative-leaf Web Functional Dependencies

Identifying negative-leaf web FDs entails identifying pairs of hyperlink labels where none of the leaves classified under paths involving one label are classified under paths involving the other. Identifying negative-leaf web FDs involves answering the question: what terms in the complete set of terms, $TE(D)$, do not lie along paths from the root of D to its leaves which classify any of the leaves reachable by paths involving a hyperlink labeled with a particular term? An example of such a web FD in KBB is 'Honda $\rightarrow \neg$ Toyota.' Unlike the discussion of negative-path web FDs, this type of web FD must be interpreted as 'none of the leaves classified by paths involving the term 'House' are classified by the paths involving the term 'Senior seat." *Mine negative-leaf-FD set* is total function $mFDs_{nl} : (\mathcal{D} \times \mathcal{T}) \rightarrow P(T)$ which given D and a term $t_i \in T = TE(D)$ returns a term-co-occurrence set. It is defined as

$$mFDs_{nl}(D,\ t_i) = TE(D) - \{t_i\} - BLTC(D,\ t_i).$$

A *negative-leaf-FD set* of a DAG D is any term-co-occurrence set returned from $mFDs_{nl}$. The complete set of negative-leaf web FDs satisfied by the DAG in Fig. 6 (left) is

$$\begin{array}{rcl}
\{\text{Computer Science} & \rightarrow & \neg\ \text{Green}, \\
\text{Mathematics} & \rightarrow & \neg\ \text{Ramakrishnan}, \\
\text{Ramakrishnan} & \rightarrow & \neg\ \text{Mathematics}, \\
\text{Ramakrishnan} & \rightarrow & \neg\ \text{Adjerid}, \\
\text{Ramakrishnan} & \rightarrow & \neg\ \text{Beattie}, \\
\text{Ramakrishnan} & \rightarrow & \neg\ \text{Green}, \\
\text{Adjerid} & \rightarrow & \neg\ \text{Ramakrishnan}, \\
\text{Adjerid} & \rightarrow & \neg\ \text{Green}, \\
\text{Beattie} & \rightarrow & \neg\ \text{Ramakrishnan}, \\
\text{Beattie} & \rightarrow & \neg\ \text{Green}, \\
\text{Green} & \rightarrow & \neg\ \text{Computer Science}, \\
\text{Green} & \rightarrow & \neg\ \text{Ramakrishnan}, \\
\text{Green} & \rightarrow & \neg\ \text{Adjerid}, \\
\text{Green} & \rightarrow & \neg\ \text{Beattie}\}.
\end{array}$$

The complete set of negative-leaf web FDs satisfied by the DAG D in Fig. 6 (right) is (the complete set of negative-path web FDs satisfied by the D)

$$\begin{array}{rcl}
\{\text{California} & \rightarrow & \neg\ \text{Virginia}, \\
\text{Virginia} & \rightarrow & \neg\ \text{California}, \\
\text{House} & \rightarrow & \neg\ \text{Senate}, \\
\text{Senate} & \rightarrow & \neg\ \text{House}\}.
\end{array}$$

Note that the $\neg$ symbol is always on the rhs of a negative web FD. Notice further that negative (-path and -leaf) and positive (-path and -leaf) web

FDs are not complements of each other, i.e., the presence of $x \rightarrow y$ does not necessarily imply the presence of $x \rightarrow \neg \{TE(D) - y\}$. Similarly, a site which satisfies $x \rightarrow \neg y$ may not satisfy $x \rightarrow \{TE(D) - y\}$.

Lemma 6.4. *Any negative web FD $x \rightarrow \neg y$ considered here also implies $y \rightarrow \neg x$.*

Lemma 6.5. *Any negative-leaf web FD in a DAG D is a negative-path web FD in D.*

Notice that the claim in Lemma 6.5 is the contrapositive of the claim in Lemma 6.2.

Corollary 6. *Any negative-leaf-FD set wrt a term t_i in D is a subset of the negative-path-FD set wrt t_i in D. In other words, $mFDs_{nl}(D,\ t_i) \subseteq mFDs_{np}(D,\ t_i)$.*

We leave it to the reader to convince thyself that the converse of Lemma 6.5 (i.e., any negative-path web FD is a negative-leaf web FD) is not true.

Lemma 6.6. *If a DAG D is a tree, then any negative-path web FD in D is a negative-leaf web FD in D.*

Corollary 7. *If a DAG D is a tree, then any negative-path-FD set wrt a term t_i in D equals the negative-leaf-FD set wrt t_i in D. In other words, if D is a tree, then $mFDs_{np}(D,\ t_i) = mFDs_{nl}(D,\ t_i)$.*

Corollary 8. *If a DAG D is a tree, then the complete set of negative-path web FDs in D equals the complete set of negative-leaf web FDs in D.*

Sketches of the proofs of Lemmas 6.2–6.6 are given in (Perugini, 2004, Ch. 4). The proof of Corollary 3 follows directly from Lemma 6.2. The proof of Corollary 4 follows directly from Lemmas 6.2 and 6.3, and the proof of Corollary 5 follows directly from Corollary 4. The proof of Corollary 6 follows directly from Lemma 6.5, and the proof of Corollary 7 follows directly from Lemmas 6.5 and 6.6.

Any DAG without an interaction set containing all terms from the site it model satisfies negative web FDs (Perugini, 2009). Moreover, any DAG with more than one distinct term satisfies either positive or negative web FDs or both (Perugini, 2009).

We have developed algorithms for mining web FDs as well as a theory of reasoning with web FDs. We have used the algorithm and theory to mine web FDs from a variety of websites to demonstrate that web hierarchies are teeming with dependencies reflecting the nature of the underlying domain and how they can be exploited in a user information-seeking interface for automatic query expansion (Perugini and Ramakrishnan, 2007).

7. Mining Web Functional Dependencies by Program Transformation

Interestingly, the classes of web FDs defined above not only can be exploited by program transformation techniques, but also can be mined through program transformation.

The following program transformation technique can mine a positive-path-FD set wrt a particular term:

$$[\![\texttt{ppfd}]\!][\mathtt{P_D},\ \texttt{input}] = \bigcup_{\mathtt{t_i}\ \in\ [\![\texttt{bptc}]\!][\mathtt{P_D},\ \texttt{input}]} \{\mathtt{t_i}\},$$

$$\text{if } ([\![\texttt{zoom}]\!][\mathtt{P_D},\ \texttt{input}]\ \cap\ [\![\texttt{zoom}]\!][\mathtt{P_D},\ \mathtt{t_i}]) = [\![\texttt{zoom}]\!][\mathtt{P_D},\ \texttt{input}],$$

where $[\![\texttt{bptc}]\!] = [\![\texttt{te}]\!][[\![\texttt{sp}]\!][\mathtt{P_D},\ \texttt{input}]] - \texttt{input}$ and is the program transformation technique we associate with $BPTC$. Notice that the above expression closely reflects the textual definition of a positive-path web FD. Intuitively, this program transformation technique mines positive-path web FDs of the form $input \rightarrow t_i$. We can mine a positive-leaf-FD set analogously:

$$[\![\texttt{plfd}]\!][\mathtt{P_D},\ \texttt{input}] = \bigcup_{\mathtt{t_i}\ \in\ [\![\texttt{bltc}]\!][\mathtt{P_D},\ \texttt{input}]} \{\mathtt{t_i}\},$$

$$\text{if } ([\![\texttt{zoom}]\!][\mathtt{P_D},\ \texttt{input}]\ \cap\ [\![\texttt{zoom}]\!][\mathtt{P_D},\ \mathtt{t_i}]) = [\![\texttt{zoom}]\!][\mathtt{P_D},\ \texttt{input}],$$

where $[\![\texttt{bltc}]\!] = [\![\texttt{te}]\!][[\![\texttt{zoom}]\!][\mathtt{P_D},\ \texttt{input}]] - \texttt{input}$ and is the program transformation technique we associate with $BLTC$. Similarly, this program transformation technique mines positive-leaf web FDs of the form $input \rightarrow t_i$. Since identifying negative web FDs does not involve containment, program transformation techniques for doing so are more simplistic than those for mining positive web FDs. Mining a negative-path-FD set entails using $[\![\texttt{te}]\!]$ to extract all of the conditional variables from the programmatic complement of $[\![\texttt{sp}]\!][\texttt{P}_\texttt{D},\ \texttt{input}]$:

$$[\![\texttt{npfd}]\!][\texttt{P}_\texttt{D},\ \texttt{input}] = [\![\texttt{te}]\!][\texttt{P}_\texttt{D} - [\![\texttt{sp}]\!][\texttt{P}_\texttt{D},\ \texttt{input}]],$$

where the $-$ symbol (i.e., minus sign) means complement. Similarly, discovering a negative-leaf-FD set entails using $[\![\texttt{te}]\!]$ to extract all of the conditional variables from the complement of $[\![\texttt{zoom}]\!][\texttt{P}_\texttt{D},\ \texttt{input}]$:

$$[\![\texttt{nlfd}]\!][\texttt{P}_\texttt{D},\ \texttt{input}] = [\![\texttt{te}]\!][\texttt{P}_\texttt{D} - [\![\texttt{zoom}]\!][\texttt{P}_\texttt{D},\ \texttt{input}]].$$

8. A Duality in Uses of Program Slicing

We have used partial evaluation and program slicing primarily as pruning operators. There is a relationship and tradeoff between these two program transformations in the context of our research. Specifically, one can think of program slicing as a transformation for

1. directly pruning a website, or
2. extracting information about what to prune from the website and partially evaluating this information away to conduct the same site pruning in (1).

Table 7 (second row) illustrates how program slicing can play role (1) above to realize interpretation 1 of out-of-turn interaction. In the previous section, we show that program slicing can also be used to mine web FDs. The members of the negative-leaf-FD set wrt some relevant term t label only the edges to be pruned from a DAG model of a website when the user communicates term t out-of-turn. Therefore, partially evaluating wrt each variable modeling each term in the negative-leaf-FD set of a particular term set to zero is sufficient to prune the undesired paths from a website to realize interpretation 1 of out-of-turn interaction.

The following expression captures these two methods of realizing interpretation 1 of out-of-turn interaction and also captures the dual role played by program slicing, i.e., role (1) above is played by $[\![\texttt{zoom}]\!]$ on the lhs and role (2) above is played by $[\![\texttt{nlfd}]\!]$ on the rhs:

$$[\![\texttt{mix}]\!][[\![\texttt{zoom}]\!][\texttt{P}_\texttt{D},\ \texttt{input}], \texttt{input} = \texttt{1}] \equiv [\![\texttt{mix}]\!][\texttt{P}_\texttt{D},\ [[\![\texttt{nlfd}]\!]\ [\texttt{P}_\texttt{D},\ \texttt{input}]] = \texttt{0},\ \texttt{input} = \texttt{1}],$$

where the notation

$$[\llbracket \texttt{nlfd} \rrbracket \ [\texttt{P}_\texttt{D}, \ \texttt{input}]] = \texttt{0},$$

denotes 'the assignment of zero to each variable modeling each member of the negative-leaf-FD set of the term modeled by the variable `input`.'

We use the word *sufficient* above, rather than *necessary*. This is because all terms labeling edges along the tails of paths to prune, when the user communicates term t out-of-turn, are members of t's negative-leaf-FD set. However, to remove that tail, we need not partially evaluate wrt all variables modeling the terms along that tail set to zero. We need only partially evaluate wrt the variable modeling the term with the minimum depth set to zero. Such a partial evaluation removes the remainder of the path by default. Intuitively, this means that we need only consider the members of the negative-leaf-FD which have the minimum depth in each path through a DAG. Considering this optimization compromises the clarity of the equivalence which we have just outlined as well as the duality in uses of program slicing. Therefore, we do not incorporate it into our presentation and expression. Also notice that when dealing with trees, we can replace $\llbracket \texttt{nlfd} \rrbracket$ with $\llbracket \texttt{npfd} \rrbracket$ in the above expression by Corollary 7.

Now let us consider how we might develop an analogous equivalence to the program transformation technique given in Table 7 (third row) for interpretation 2 of out-of-turn interaction. Recall to support interpretation 2 we replace $\llbracket \texttt{zoom} \rrbracket$ with $\llbracket \texttt{sp} \rrbracket$ on the lhs of the above equivalence expression. Thus, it is natural to consider replacing $\llbracket \texttt{nlfd} \rrbracket$ with $\llbracket \texttt{npfd} \rrbracket$ on the rhs of the above expression to form an analogous equivalence expression for interpretation 2 of out-of-turn interaction. However, upon careful examination we see that the rhs is not equal to the lhs and, thus, does not realize interpretation 2 of out-of-turn interaction. This is because if $\texttt{P}_\texttt{D}$ models a non-tree DAG, then our use of partial evaluation, as described above, to remove undesired paths may also remove leaves which lie at the end of desired paths! For example, consider partially evaluating a programmatic representation of the website illustrated in Fig. 6 (left) wrt the negative-path-FD set of 'Adjerid.' This transformation removes the source (leaf) of the edge labeled 'Adjerid' which is unfaithful to interpretation two. However, the equivalence expression for interpretation 1 can serve as a surrogate equivalence expression for interpretation 2 of out-of-turn interaction with trees (by Corollary 4), where $\llbracket \texttt{zoom} \rrbracket$

and $[\![\texttt{nlfd}]\!]$ can be replaced by $[\![\texttt{sp}]\!]$ and $[\![\texttt{npfd}]\!]$, respectively. We are unable to develop equivalence expressions for the generalizations of these interpretations of out-of-turn interaction here due to their template-oriented nature, i.e., the presence of $[\![\texttt{SL}]\!]$.

Studying this duality reveals that there might be simpler methods for discerning which branches must be removed in specialized DAG classes. The following program transformation technique realizes out-of-turn interaction, but is specifically tailored toward taking advantage of the mutually exclusive property in DAGs and trees:

$$[\![\texttt{dead—code}]\!][[\![\texttt{mix}]\!][\texttt{P}_\texttt{D},\ [[\![\texttt{tle}]\!][\texttt{P}_\texttt{D},\ \texttt{input}] - \texttt{input}] = \texttt{0},\ \texttt{input} = \texttt{1}]],$$

where $[\![\texttt{dead—end}]\!]$ represents a dead-code detection and elimination transformation (Chen et al., 2001; Wegman and Zadeck, 1991). Static slicing methods can detect dead-code (Bergeretti and Carré, 1985). Detection is typically conducted in a debugging context since such statements are usually unexecutable due to the presence of a bug (Tip, 1995). This approach illustrates that partial evaluation is a specialization of program slicing wrt programmatic representations of levelwise, mutually exclusive DAG models of websites. Here $[\![\texttt{tle}]\!]$ supports TLE programmatically and involves extracting all the structural variables at a particular level of nesting. It is intended to be polymorphic in that its input can be either given extensionally as a level number or intensionally as a term which occurs at the desired level. Notice that the latter usage is unambiguous only in the case of mutually exclusive classes (which are the only classes it is used for here).

Other researchers have echoed similar connections between partial evaluation and slicing; the two techniques have been shown to yield similar results in some situations and different results in others (Reps and Turnidge, 1996).

9. Program Transformations for Supplemental Personalized Interactions

To target our formal model for representing and reasoning about personalized interaction in hierarchical websites, we illustrate how the formalisms can be augmented to support supplementary personalized interactions. Moreover, the success of a personalization system relies on those finer touches which deliver a compelling experience to the end-user. Studies of out-of-turn interaction have revealed that users desire supplemental interactions to enhance the personalized experience while interacting with hierarchical hypermedia (Perugini, 2004, Ch. 6). We showcase a suite of such interactions and relate them to program transformation techniques.

Meta-enquery: What May I Say?

To employ an interactive information system effectively, users need a mechanism to stay abreast of its underlying vocabulary during the interaction. Keeping users attuned to the communicable information is an issue all information systems must address. Yankelovich echoes this issue as ‘how do users know what to say?’ (Yankelovich, 1996). While each hyperlink label is always available, a new (or causal) user of a site may be unfamiliar with or unaware of subsequent solicitations for input and, hence, the terms of information seeking the designer has modeled which are available to supply out-of-turn. This is a classic problem in IR research, has been identified and described by many (Bodner and Chignell, 1999; Croft and Thompson, 1987; Marchionini, 1997; Sacco, 2000; Williams, 1984), and is endemic to all IR systems. Users typically have a ‘limited knowledge of a given database’ (Williams, 1984) and thus experience ‘difficulty expressing their information need’ (Bodner and Chignell, 1999).

The *naive what may I say?* interaction permits the user to determine what partial information remains unspecified thus far in her interaction with a system. It can be trivially supported by TE. Supporting this interaction programmatically entails using $[\![\texttt{te}]\!]$ to extract all unique structural variables, which correspond to the partial information available to commit the system, from the representation.

In a voluminous space such a set of terms may be large and overload the user and, therefore, we provide a similar interaction, called *what may I say?*, which entails clustering terms by level (facet), using TLE, to help orient users. This approach is, of course, applicable only to levelwise sites. When used in this manner, this interaction also updates the user on how their previous interactions have affected the remaining choices and, thus, provides context. Supporting this interaction programmatically entails using $[\![\texttt{tle}]\!]$ to extract all unique structural variables at a desired level of nesting from the representation.

Restructure Classification

Notice that while an out-of-turn interaction paradigm subsumes the corresponding browsing paradigm (see Lemma 4.1), interaction sequences *deriv-*

able through out-of-turn interaction are not *describable* by browsing. The *restructure classification* interaction, which is only applicable to levelwise sites, enables the creation of a personalized browsing hierarchy, which can be further navigated by browsing or out-of-turn interaction. For example, a website organized along an author-journal-title motif could be restructured into a journal-author-title organization to support interactive aggregation scenarios.

Approaches to this interaction are to permit a user to restructure an entire enumerative classification a priori or incrementally (Perugini and Ramakrishnan, 2006; Perugini, 2006). The semistructured data community (Abiteboul et al., 2000; Florescu et al., 1998) has advocated restructuring websites through declarative queries (Fernández et al., 1997, 1998). Such an approach restructures an entire organization in one-stroke. A similar approach is taken in *User-Defined Hierarchies* (Wilson and Bergeron, 1999). An alternate approach is to permit the user to define the ensuing interaction incrementally while browsing. This interaction style has been echoed metaphorically as 'magically [laying] down track to suggest useful directions to go based on where [one has] been so far and what [one is] trying to do' (Hearst, 2000). These ideas have been studied in the *Flamenco Search System Project* (Hearst et al., 2002). *Flamenco* explores faceted classification in various catalogs and websites. The adaptive hypermedia community (Bodner and Chignell, 1999; Brusilovsky, 2001) is a proponent of such an incremental approach to classification specification. In adaptive hypermedia, links are dynamic (Pokorny, 2001) and lead to different destinations for different users.

Counterintuitively, our nested-conditional representation of interaction need not be restructured at all to accommodate this interaction! A method to extract structural variables corresponding to a particular level of conditional nesting is sufficient. These variables can be used to create hyperlinks on a generated webpage, each initiating an interaction which appears to be browsing, but is actually out-of-turn interaction wrt the program representation. This, yet again, emphasizes the importance of investing in representation, and the flexibility in our programmatic representations.

Thus, support for this interaction requires level-order, edge-label extraction, TLE. Supporting this interaction programmatically entails the same code extraction used for automatic query expansion in mutually exclusive classes of hierarchical hypermedia, i.e., $[\![\texttt{tle}]\!]$. Notice that the edge-label extraction for this interaction is wrt one particular level and is, thus, a specialization of the edge-label extraction required for (*naive*) *what may I say?*

which is wrt all levels. In other words, the terms returned from TLE are a subset of those returned from TE.

Collect Results

This is a termination interaction which allows the user to request that the classification be flattened and re-presented as a flat list of hyperlinks to relevant content pages. For instance, while drilling-down a hierarchical website a user may wish to curb further interaction at a particular point and retrieve a flat list of the terminal webpage reachable from that point. At such a point, the user may not care to pursue further distinctions or dependencies. Effectively, the user has prematurely declared that the interaction is over. This interaction involves the forward-propagation used to support out-of-turn interaction and is defined by CR introduced above. We support CR programmatically with the $[\![\texttt{cr}]\!] = [\![\texttt{forward}]\!][\texttt{P}_\texttt{D}, \texttt{ page}]$ program transformation technique – a specialization of $[\![\texttt{zoom}]\!]$ which only extracts terminal variables from the representation. Notice that *collect results* suggests the need to invert the factoring of terminal information which we built into the programmatic representation with nesting. Observe that the function which supports *collect results* also can be used to generate *information previews* — hyperlink annotations, typically parenthesized and containing a frequency count of the documents reachable from the hyperlink it decorates — which have been shown important in situations where the user is confronted with a decision regarding where to go next (Plaisant et al., 1999). Such previews are now a de facto standard in most faceted browsing and search systems and web directories.

Inverse Personalization

Inverse personalization is so named not because its goals run contrary to personalization, but because in a given representation it conducts a mapping from terminal (e.g., leaf pages) to structural information (e.g., hyperlink labels). This interaction helps support situations where the user knows what information she wants, but is interested in determining how to retrieve that information and, in this manner, permits users to conduct 'what if' analyzes. For instance, using *inverse personalization* with an online apartment recommender system, a user can issue a request such as 'under what conditions will the Georgetown apartments be the only choice?' Such a user is interested in the terms along the interaction sequences leading to that terminal information (e.g., if you want a swimming pool, covered parking, and free wi-fi). Such

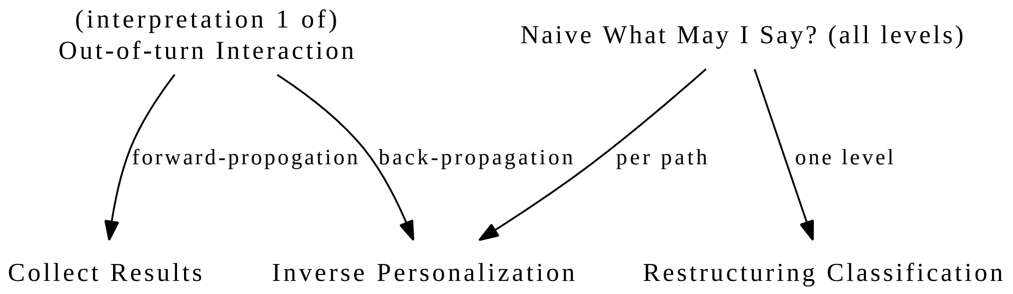


Figure 8: A taxonomy of supplementary interactions. Directed arrows represent specialization relations.

Table 9: Program transformation techniques for and observations on the supplemental personalized interactions. (legend: Sup Int = Supplemental interaction, Ind = Independent, Pgm Trans Tech = Program transformation technique, NWhat? = *naive what may I say?*, What? = *what may I say?*, RC = *restructuring classification*, CR = *collect results*, and P^{-1} = *inverse personalization*).

Sup Int	Read-only	Closed	Ind	Pgm Trans Tech
NWhat?	$\surd$	$\surd$	$\times$	$\cup\ \{[\![\texttt{te}]\!]\ [\texttt{P}_\texttt{D},\ \texttt{x}]\}$, for every structural variable `x`
What?	$\surd$	$\surd$	$\times$	$\cup\ \{[\![\texttt{tle}]\!]\ [\texttt{P}_\texttt{D},\ \texttt{x}]\}$, for every level `x`
RC	$\surd$	$\surd$	$\surd$	$[\![\texttt{tle}]\!]\ [\texttt{P}_\texttt{D},\ \texttt{x}]$, for one level `x`
CR	$\times$	$\times$	$\surd$	$[\![\texttt{forward}]\!]\ [\texttt{P}_\texttt{D},\ \texttt{page}]$
$\mathbf{P^{-1}}$	$\surd$	$\surd$	$\surd$	$[\![\texttt{te}]\!]\ [[\![\texttt{sp}]\!]\ [[\![\texttt{backward}]\!]\ [[\![\texttt{SL}]\!]\ [\texttt{P}_\texttt{D},\ \texttt{input}]]]]$

personalized interactions are supported in our approach by back-propagating from the leaf webpages returned by SL and optionally extracting each term in each path from the root as a whole, using TE, or per interaction sequence, using $TE \circ SP$.

Inverse personalization is similar to generalized interpretation 1 of out-of-turn interaction, as evidenced by its use of SL. However, akin to *collect results*, this interaction is a specialized form of out-of-turn interaction. Honing in on leaves (i.e., forward-propagation) is unnecessary; back-propagation is sufficient and, therefore, defines this interaction. In a nested conditional representation, terminal information is indexed by a terminal variable (e.g., `page`) which is not user-modifiable (as is structural information, e.g., `software`). Therefore, we need a transformation capable of exploiting terminal variables such as backward slicing. Collecting the terms along the resulting paths requires an extraction similar to that required for *naive what may I say?*, i.e., $[\![\texttt{te}]\!]$.

Fig. 8 illustrates the relationships between these supplemental personalized interactions.

Table 9 contains program transformation techniques for these supplemental personalized interactions. Notice that the *what may I say?* and *restructuring classification* interactions require slight variations of the same transformation technique. Since *collect results* and *inverse personalization* are each specializations of out-of-turn interaction, they relate to forms of forward and backward slicing, respectively (and code extraction) and, thus, when combined come close to realizing out-of-turn interaction.

Table 9 also summarizes our observations on these personalized interactions. The top row of the matrix lists attributes of the interactions listed on the far left. A read-only interaction is one which only manipulates the representation rather than modifying it. An interaction is closed if it accepts a representation and returns a representation and, thus, does not curb further interaction. Independent interactions are those which not only complement out-of-turn interaction, but are also defined, applicable, and useful in its absence.

10. Related Research

Three main ideas underly the work presented here:

1. explicit modeling of interaction,
2. capture of partial information,
3. use of program transformations.

Any information systems context where one or more of the above ideas apply is ground for applying the techniques presented in this article.

Modeling interaction is has been recognized as important in multiple interactive information retrieval projects, especially to manage interaction and maintain state. Belkin et al. (1995) introduced the idea of an *interaction script* which can be thought of as a program for interaction, though expressed in English rather than program codes, and is intended to be interpreted. Graunke et al. (2001) describe an approach to automatically restructure batch programs for interactive use on the web. An important issue addressed is maintaining state across web interactions which use the stateless HTTP protocol. The approach involves capturing first-class continuations (Friedman et al., 2001) through the `call/cc` (call-with-current-continuation) facility in Scheme. Since first-class continuations can be saved and resumed, they are an ideal construct for capturing and restoring state between user interactions over the web. Using a similar idea based on continuations, Queinnec (2000) 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. Moreover, Quan et al. (2003) use continuations and currying (Ullman, 1997) with explicit representations to postpone, save, and resume interactions with intrusive dialog boxes, including partially-filled boxes, 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 these efforts in its focus on out-of-turn interactions. Nevertheless, these projects reinforce our viewpoint that information system design can benefit from investing in representations of interaction, and exploring concepts from programming languages.

Until recently, the issue of context in information systems has received comparatively little attention. With the proliferation of mobile environments and information appliances (Bergman, 2000), coupled with an improved understanding of their usages (Perry et al., 2001), capturing and reusing context has become important. Context can be viewed as a rich form of partial information, allowing the work presented here to be applied toward exploiting it. This property is attractive because the partial information can potentially be multi-faceted – information about a user's preferences (e.g., in a recommender system), a user's location (e.g., in a mobile environment; Hightower and Borriello, 2001), or a user's partially completed interaction (e.g., a shopping cart at Amazon). The representations studied in this article can be generalized to accommodate such richer forms of partial information.

Finally, the use of program transformations as presented here is relatable to the larger community which aims to systematize and automate the software engineering of complex, interactive web applications (Graunke et al., 2001). It finds relevance in many website re-structuring and re-engineering efforts (Ricca et al., 2001; Ricca and Tonella, 2001a,b; Ricca et al., 2002), especially in the adaptive and semantic web contexts, declarative site specification (Fernández et al., 1998), and improving site usability (Spiliopoulou, 2000). Work (Gonçalves et al., 2001; Perugini et al., 2004a) has also been done to incorporate program transformations for personalization into models for digital libraries, e.g., 5S (Gonçalves et al., 2004). While there are established and effective models for classical information retrieval, e.g., vector-space (Baeza-Yates and Ribeiro-Neto, 1999), models for solutions to information personalization problems are in their infancy. We believe that program transformations, which are under-explored in the personalization community, suggest helpful metaphors for developing such models.

11. Discussion

We implemented the graph-theoretic transformations described here in ML (*Meta*-Language) using SML/NJ (Ullman, 1997) and explored the feasibility of the website – program associations by conducting verification experiments on programmatic representations of interaction with PVS, ODP, and the Virginia Tech online timetable of courses using program transformation software systems (Narayan et al., 2004). The *set calculator* of *CodeSurfer* (Anderson et al., 2003), a program slicing system, was especially helpful for performing set-theoretic operations over slices. We discuss the details of the program transformation systems we used in (Perugini, 2004, appendix C).

11.1. Putting It All Together

We have developed a tool and markup language for automatically generating web user interfaces with support for a combination of the supplemental personalized interactions introduced in Section 9 (Perugini et al., 2004b). Moreover, the following three software components, and an interaction manager for coordination, constitute a customizable software framework for creating web personalization systems which support mixed-initiative interaction (Narayan et al., 2004):

- **Web Transformation Engine**: a web service which prunes a hierarchical website when given out-of-turn input. The engine is based on the program transformation techniques in Table 7 and implements a form of forward followed by backward program slicing using XSLT. Since it handles in-turn and out-of-turn inputs (captured by interaction interfaces below) in a uniform manner, it enables out-of-turn interaction without requiring the site designer to anticipate the points at which out-of-turn interaction can happen. Using a single transformation technique this engine supports out-of-turn interaction with both faceted and unfaceted sites.

- ***Speech UI***: a voice interface, implemented with VoiceXML and X+V, which permits the user to supply out-of-turn inputs through speech and enables multimodal interaction when used in conjunction with hyperlinks.

- ***Extempore***: a cross-platform toolbar plugin embedded into the Mozilla Firefox web browser; the analog of the speech user interface using a textual modality.

Extempore (Perugini and Ramakrishnan, 2003b) and a variety of alternate interaction interfaces (Perugini and Ramakrishnan, 2006; Perugini, 2006) are are available for demonstration from our group's webpage at `http://oot.cps.udayton.edu`. We have instantiated this framework for the following case studies which afford interactions initiated by these interfaces, coordinated by this interaction manager, and staged by this transformation engine:

- GAMS (Guide to Available Mathematical Software) at `http://gams.nist.gov` (Perugini et al., 2000; Ramakrishnan and Perugini, 2001),

- Project Vote Smart at `http://votesmart.org` (Narayan et al., 2004; Perugini and Ramakrishnan, 2003b),

- Pigments through the Ages at `http://webexhibits.org/pigments/`,

- CITIDEL (Computing and Information Technology Interactive Digital Educational Library) at `http://citidel.org` (Perugini et al., 2004a,b),

- Open Directory Project at `http://dmoz.org` (Narayan et al., 2004), and

- Online Virginia Tech Timetables of Classes accessible through `http://vt.edu` (Perugini and Ramakrishnan, 2007).

These sites are an assortment of the classes of hierarchical hypermedia introduced in Section 5.

11.2. Contributions

Our research, and specifically out-of-turn interaction, is part of a larger research effort which seeks to marry navigational (e.g., the *Yahoo!* directory) and direct search (e.g., *Google*) (Sacco and Tzitzikas, 2009). Our research has made the following contributions to the interactive information retrieval, personalization, dialog management, and programming languages communities.

- **New approach to personalization**: We applied program transformations, seemingly unrelated techniques predominately used in compilers and debuggers, to a problem in interactive information retrieval. While viewing a website as a program and website transformation for personalization as program transformation is unorthodox, it offers a new way of thinking about personalized interaction, especially with hierarchical hypermedia. While others have studied personalization in information hierarchies using more traditional approaches, e.g., Dalamagas et al. (2007) take a data mining approach to this problem, to the best of our knowledge, no one has taken a program transformation approach to personalizing interaction in hierarchical domains. This new approach lead to the following contributions.

- **Unifies search in faceted and unfaceted domains**: Our nested conditionals representation of hierarchical hypermedia can be used to model faceted sites (e.g., PVS) and unfaceted sites (e.g., ODP). As a result, a single program transformation technique, e.g., any technique in Table 7, can be used to personalize interaction in both domains and, thereby, unifies search with two fundamentally distinct forms of hypermedia. Currently, there are toolkits available for building faceted browsing and search applications (e.g., *Flamenco* at `http://flamenco.berkeley.edu`). However, search over unfaceted spaces such as the *Yahoo!* directory or ODP is different. We can personalize both spaces with the same transformation using our approach.

- **New line of dialog management research**: It is important to note that our research aims to model a user's *interaction* with a site, rather than simply a site schema. While traditional data-oriented approaches, such as XML or a relational database, might be appropriate for the latter, programming languages offer attractive constructs (e.g., booleans, conditionals, functions, `gotos`) for modeling interaction as we have demonstrated above. While it is easy to presume that modeling a user's interaction with a hierarchical website as a program is equivalent to modeling it as an XML document or in a database relation, there is an important distinction between the two. An imperative program, like the only possible style of interaction with the static, rigid, one-size-fits-all organization of an information hierarchy, has a default order of execution; there is no such analog in an XML document[3] or database relation. This distinction is significant because it compelled us to explicitly recognize orders of execution and identify alternate domains where order is relevant. One such domain is human-computer dialogs; often a dialog script also has a default order of evaluation.

Thus, our experience using programs to represent interaction with hierarchical sites helped connect our work to human-computer dialog management and, in particular, mixed-initiative interaction (Allen, 1999). Specifically, we recognized that supporting the user in interleaving in-turn and out-of-turn inputs while browsing a website is simple form of mixed-initiative interaction (Perugini and Ramakrishnan, 2003b) and were compelled to study computational models for human-computer dialogs (Haller and McRoy, 1997). This connection may not have been fostered had we approached personalized interaction from a more traditional perspective.

Dialogs are now pervasive in computer systems and used in automated teller machines and airport kiosks. While the fluidity inherent in dialog is essential to provide a natural experience to the user, it is also a vice for the implementor due to the directions in which the user might steer the dialog, which must all be captured in the implementation. These problems are difficult since dialogs range in complexity from those modeled after a simple pre-defined series of questions and answers (sometimes referred to as slot-fillers) to those which give the user a great deal of control over the direction in which to steer the dialog (Allen, 1999; Ferguson and Allen, 1998). The connection we have fostered is that

[3]We do not intend to imply that in using the program transformation approach we cannot or should not implement the transformation techniques using tools such as XML/XSLT. Rather, we are saying that we may not have made the connection to dialogs had we started out with such a database approach.

a dialog script, like a program, has a default order of execution, and applying program transformations alters that order, thereby, making the dialog flexible without having to explicitly hardcode all supported deviations from the default order within the script itself. This is a primary contribution of our research. As a result, we have also discovered that program transformations are helpful for specifying dialogs (Capra et al., 2003). Putting personalized web interaction on a fundamentally different landscape (i.e., program transformation) gave birth to this new line of research (Ramakrishnan et al., 2002; Capra et al., 2003; Perugini et al., 2007) which may not have come to fruition had we started with more traditional approaches (XML, databases).

- **Technology assimilation by reduction to theoretical principals**: Using the ideas described here, many relevant interactive applications and dialog standards can be re-visited and re-studied by their support for personalizing interaction. For instance, consider the VoiceXML markup language designed to simplify the construction of interactive voice response systems (McGlashan et al., 2001). VoiceXML markup tags describe prompts, forms, and fields which constitute a dialog, and support both directed and mixed-initiative dialogs. Ramakrishnan, Capra, and Pérez-Quiñones (2002) have shown that VoiceXML's form-interpretation algorithm is a partial evaluator in disguise. Such connections are noteworthy because they reduce rapidly emerging and constantly evolving technologies to fundamental theoretical operations and, therefore, improve assimilation of such technologies.

- **Towards a model for information personalization**: Our use of program transformations casts personalization in a formal setting and provides a systematic and implementation-neutral approach to designing information personalization systems, especially those in hierarchical hypermedia domains with support for mixed-initiative interaction. Moreover, flexibility is built into the approach. When a given program (representation, transformation) pair is deemed inadequate to achieve the desired form of personalization, we have two choices: keep the representation fixed and investigate or design alternative program transformation techniques capable of realizing the desired interaction or, alternatively, develop representations wrt a fixed suite of transformation techniques. Conversely, we can develop new forms of personalization by creatively applying program transformations to representation. This ability to vary the elements of a model not only suggests the extensibility of the methodology, but also constitutes the creativity in our research.

11.3. Future Work

The interpretations for out-of-turn interaction presented here are not exhaustive, especially since we defined open-ended generalized interpretations of out-of-turn interaction. For instance, we might expand the scope of addressable out-of-turn information by modeling the terms on the leaf webpages (i.e., terminal information) and making them available as out-of-turn input for the user to supply. In such case, we might implement SL using a text-based search engine. Nevertheless, use of a general program transformation, such as slicing, suggests that alternate interpretations also can be accommodated. For instance, we intend to explore additional program transformations, such as generalized partial evaluation (Futamura et al., 1991; Takano, 1991) and parameterized partial evaluation (Consel and Khoo, 1993), as well as other related programming language concepts, including reflection (Maes, 1987), concept assignment (Harman et al., 2002), and program schemas (Ianov, 1960), and study the personalized interactions these techniques might enable; see (Perugini, 2004, appendix B) for more information.

We also intend to develop a computational model, based on first-class closures and continuations, for simplifying the implementation of mixed-initiative human-computer dialogs, and develop algorithms for automatically generating an optimal dialog script from a high-level specification of a dialog. While first-class closures and continuations are powerful programming constructs, they are under-utilized outside of compilers and interpreters.

Our long term research goal is to formalize and simplify the design and implementation of dialog-based systems, especially those involving mixed-initiative human-computer interactions. The approach involves empirically studying user interactions with systems and formally casting those interactions through a programming languages lens.

Acknowledgments

We thank Travis Suel at the University of Dayton for re-designing and -implementing the out-of-turn framework and multiple interface interfaces, including *Extempore*, and for his design and implementation of new interfaces. John Cresencia at the University of Dayton also provided software support in a variety of ways.

Vitae

Saverio Perugini is an Assistant Professor in the Department of Computer Science at the University of Dayton. His research interests include information personalization, web mining, and functional programming. He has a Ph.D. in Computer Science from Virginia Tech.

References

Abiteboul, S., Buneman, P., Suciu, D., 2000. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, San Francisco, CA.

Allen, J., 1999. Mixed-Initiative Interaction. IEEE Intelligent Systems 14 (5), 14–16.

Anderson, P., Reps, T., Teitelbaum, T., 2003. Design and Implementation of a Fine-Grained Software Inspection Tool. IEEE Transactions on Software Engineering 29 (8), 721–733.

Baeza-Yates, R., Ribeiro-Neto, B., 1999. Modern Information Retrieval. ACM Press, New York, New.

Belkin, N., Cool, C., Stein, A., Thiel, U., 1995. Cases, Scripts, and Information-Seeking Strategies: On the Design of Interactive Information Retrieval Systems. Expert Systems with Applications 9 (3), 379–395.

Bergeretti, J.-F., Carré, B., 1985. Information-Flow and Data-Flow Analysis of While-Programs. ACM Transactions on Programming Languages and Systems 7 (1), 37–61.

Bergman, E. (Ed.), 2000. Information Appliances and Beyond. The Morgan Kaufmann Series on Interactive Technologies. Morgan Kaufmann, San Francisco, CA.

Binkley, D., Gallagher, K., 1996. Program Slicing. In: Zelkowitz, M. (Ed.), Advances in Computers. Vol. 43. Academic Press, Amsterdam, pp. 1–50.

Bodner, R., Chignell, M., 1999. Dynamic Hypertext: Querying and Linking. ACM Computing Surveys 31 (4es), article No. 15.

Brusilovsky, P., 2001. Adaptive Hypermedia. User Modeling and User-Adapted Interaction 11 (1–2), 87–110.

Capra, R., Narayan, M., Perugini, S., Ramakrishnan, N., Pérez-Quiñones, M., 2003. The Staging Transformation Approach to Mixing Initiative. In: Tecuci, G. (Ed.), Working Notes of the IJCAI 2003 Workshop on Mixed-Initiative Intelligent Systems. AAAI/MIT Press, Menlo Park, CA, pp. 23–29.

Chen, Z., Xu, B., Yang, H., 2001. Detecting Dead Statements for Concurrent Programs. In: Proceedings of the International Conference on Software Maintenance (SCAM). IEEE Computer Society, Los Alamitos, CA, pp. 67–72.

Consel, C., Khoo, S., 1993. Parameterized Partial Evaluation. ACM Transactions on Programming Languages and Systems 15 (3), 463–493.

Croft, W., Thompson, R., 1987. I^{3}R: A New Approach to the Design of Document Retrieval Systems. Journal of the American Society for Information Science 38 (6), 389–404.

Dalamagas, T., Bouros, P., Galanis, T., Eirinaki, M., Sellis, T., 2007. Mining User Navigation Patterns for Personalizing Topic Directories. In: Fundulaki, I., Polyzotis, N. (Eds.), Proceedings of the Ninth Annual ACM International Workshop on Web Information and Data Management (WIDM). ACM Press, New York, NY, pp. 81–88.

Dhyani, D., NG, W., Bhowmick, S., 2002. A Survey of Web Metrics. ACM Computing Surveys 34 (4), 469–503.

Eirinaki, M., Vazirgiannis, M., 2003. Web Mining for Web Personalization. ACM Transactions on Internet Technology 3 (1), 1–27.

Ferguson, G., Allen, J., 1998. TRIPS: An Integrated Intelligent Problem-solving Assistant. In: Mostow, J., Rich, C., Buchanan, B. (Eds.), Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI). AAAI Press, Menlo Park, CA, pp. 567–572.

Fernández, M., Florescu, D., Kang, J., Levy, A., Suciu, D., 1998. Catching the Boat with Strudel: Experiences with a Web-Site Management System. ACM SIGMOD Record 27 (2), 414–425.

Fernández, M., Florescu, D., Levy, A., Suciu, D., 1997. A Query Language for a Web-Site Management System. SIGMOD Record 26 (3), 4–11.

Florescu, D., Levy, A., Mendelzon, A., 1998. Database Techniques for the World-Wide Web: A Survey. SIGMOD Record 27 (3), 59–74.

Friedman, D., Wand, M., Haynes, C., 2001. Essentials of Programming Languages, Second Edition. MIT Press, Cambridge, MA.

Futamura, Y., Nogi, K., Takano, A., 1991. Essence of Generalized Partial Computation. Theoretical Computer Science 90 (1), 61–79.

Gonçalves, M., Fox, E., Watson, L., Kipp, N., 2004. Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries. ACM Transactions on Information Systems 22 (2), 270–312.

Gonçalves, M., Zafer, A., Ramakrishnan, N., Fox, E., 2001. Modeling and Building Personalized Digital Libraries with PIPE and 5SL. In: Smeaton, A., Callan, J. (Eds.), Proceedings of the Joint DELOS-NSF Workshop on Personalisation and Recommender Systems in Digital Libraries. Dublin, Ireland.

Graunke, P., Findler, R., Krishnamurthi, S., Felleisen, M., 2001. Automatically Restructuring Programs for the Web. In: Proceedings of the Sixteenth IEEE International Conference on Automated Software Engineering (ASE). IEEE Computer Society, Los Alamitos, CA, pp. 211–222.

Haller, S., McRoy, S. (Eds.), 1997. Computational Models for Mixed Initiative Interaction: Papers from the 1997 AAAI Spring Symposium. No. SS-97-04. AAAI Press, Menlo Park, CA.

Harman, M., Gold, N., Hierons, R., Binkley, D., 2002. Code Extraction Algorithms which Unify Slicing and Concept Assignment. In: Proceedings of the Ninth IEEE Working Conference on Reverse Engineering (WCRE). IEEE Computer Society, Los Alamitos, CA, pp. 11–21.

Harman, M., Hierons, R., 2001. An Overview of Program Slicing. Software Focus 2 (3), 85–92.

Hearst, M., 1999. Mixed-Initiative Interaction. IEEE Intelligent Systems 14 (5), 14.

Hearst, M., 2000. Next Generation Web Search: Setting Our Sites. IEEE Data Engineering Bulletin 23 (3), 38–48.

Hearst, M., Elliott, A., English, J., Sinha, R., Swearingen, K., Yee, K.-P., 2002. Finding the Flow in Web Site Search. Communications of the ACM 45 (9), 42–49.

Hightower, J., Borriello, G., 2001. Location Systems for Ubiquitous Computing. IEEE Computer 34 (8), 57–66.

Hildum, D., Cohen, J., 1990. A Language for Specifying Program Transformations. IEEE Transactions on Software Engineering 16 (6), 630–638.

Horwitz, S., Reps, T., Binkley, D., 1990. Interprocedural Slicing Using Dependency Graphs. ACM Transactions on Programming Languages and Systems 12 (1), 26–60.

Ianov, Y., 1960. The Logical Schemes of Algorithms. In: Problems of Cybernetics. Vol. 1. Pergamon Press, New York, NY, pp. 82–140.

Jackson, D., Rollins, E., 1994a. A New Model of Program Dependences for Reverse Engineering. ACM SIGSOFT Software Engineering Notes 19 (5), 2–10.

Jackson, D., Rollins, E., 1994b. Chopping: A Generalisation of Slicing. Tech. Rep. CMU-CS-94-169, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Jones, N., 1996. An Introduction to Partial Evaluation. ACM Computing Surveys 28 (3), 480–503.

Jones, N., 1997. Computability and Complexity from a Programming Perspective. Foundations of Computing Series. MIT Press, Cambridge, MA.

Jones, N., Gomard, C., Sestoft, P., 1993. Partial Evaluation and Automatic Program Generation. Prentice Hall International.

Lyle, J., Weiser, M., 1987. Automatic Program Bug Location by Program Slicing. In: Proceedings of the Second International Conference on Computers and Applications. IEEE Computer Society, Los Alamitos, CA, pp. 877–882.

Maes, P., 1987. Concepts and Experiments in Computational Reflection. In: Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA). ACM Press, New York, NY, pp. 147–155.

Marchionini, G., 1997. Information Seeking in Electronic Environments. Cambridge Series on Human-Computer Interaction. Cambridge University Press, Cambridge, UK.

McGlashan, S., Burnett, D., Danielsen, P., Ferrans, J., Hunt, A., Karam, G., Ladd, D., Lucas, B., Porter, B., Rehor, K., Tryphonas, S., 2001. Voice eXtensible Markup Language: VoiceXML. Tech. rep., VoiceXML Forum, version 2.0.

Mobasher, B., Cooley, R., Srivastava, J., 2000. Automatic Personalization Based on Web Usage Mining. Communications of the ACM 43 (8), 142–151.

Mulvenna, M., Anand, S., Büchner, A., 2000. Personalization on the Net using Web Mining. Communications of the ACM 43 (8), 122–125.

Narayan, M., Williams, C., Perugini, S., Ramakrishnan, N., 2004. Staging Transformations for Multimodal Web Interaction Management. In: Najork, M., Wills, C. (Eds.), Proceedings of the Thirteenth International ACM World Wide Web Conference (WWW13). ACM Press, New York, NY, pp. 212–223.

Perry, M., O'Hara, K., Sellen, A., Brown, B., Harper, R., 2001. Dealing with Mobility: Understanding Access Anytime, Anywhere. ACM Transactions on Computer-Human Interaction 8 (4), 323–347.

Perugini, S., 2004. Program transformations for information personalization. Ph.D. thesis, Department of Computer Science, Virginia Tech.

Perugini, S., 2006. Real-time Query Expansion and Procedural Interfaces for Information Hierarchies. In: Broder, A., Maarek, Y. (Eds.), Proceedings of the International ACM SIGIR Workshop on Faceted Search.

Perugini, S., 2008. Symbolic Links in the Open Directory Project. Information Processing and Management 44 (2), 910–930.

Perugini, S., 2009. Supporting Multiple Paths to Objects in Information Hierarchies: Faceted Classification, Faceted Search, and Symbolic Links. Information Processing and ManagementDOI: `10.1016/j.ipm.2009.06.007`.

Perugini, S., Anderson, T., Moroney, W., 2007. A Study of Out-of-turn Interaction in Menu-based, IVR, Voicemail Systems. In: Rosson, M., Gilmore, D. (Eds.), Proceedings of the Twenty-fifth International ACM Conference on Human Factors in Computing Systems (CHI). ACM Press, New York, NY, pp. 961–970.

Perugini, S., Lakshminarayanan, P., Ramakrishnan, N., 2000. Personalizing the GAMS Cross-Index. Tech. Rep. TR-00-01, Department of Computer Science, Virginia Tech.

Perugini, S., McDevitt, K., Richardson, R., Pérez-Quiñones, M., Shen, R., Ramakrishnan, N., Williams, C., Fox, E., 2004a. Enhancing Usability in CITIDEL: Multimodal, Multilingual, and Interactive Visualization Interfaces. In: Chen, H., Wactlar, H., Chen, C., Lim, E.-P., Christel, M. (Eds.), Proceedings of the Fourth International ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL). ACM Press, New York, NY, pp. 315–324.

Perugini, S., Ramakrishnan, N., 2003a. Personalizing Interactions with Information Systems. In: Zelkowitz, M. (Ed.), Advances in Computers. Vol. 57. Academic Press, Amsterdam, pp. 323–382.

Perugini, S., Ramakrishnan, N., 2003b. Personalizing Web Sites with Mixed-Initiative Interaction. IEEE IT Professional 5 (2), 9–15.

Perugini, S., Ramakrishnan, N., 2005. A Generative Programming Approach to Interactive Information Retrieval: Insights and Experiences. In: Glück, R., Lowry, M. (Eds.), Proceedings of the Fourth International ACM Conference on Generative Programming and Component Engineering (GPCE). Vol. LNCS 3676. Springer, Berlin, pp. 205–220.

Perugini, S., Ramakrishnan, N., 2006. Interacting with Web Hierarchies. IEEE IT Professional 8 (4), 19–28.

Perugini, S., Ramakrishnan, N., 2007. Mining Web Functional Dependencies for Flexible Information Access. Journal of the American Society for Information Science and Technology (JASIST) 58 (12), 1805–1819.

Perugini, S., Ramakrishnan, N., Fox, E., 2004b. Automatically Generating Interfaces for Personalized Interaction with Digital Libraries. Tech. Rep. cs.DL/0402022, Computing Research Repository (CoRR).

Plaisant, C., Shneiderman, B., Doan, K., Bruns, T., 1999. Interface and Data Architecture for Query Preview in Networked Information Systems. ACM Transactions on Information System 17 (3), 320–341.

Pokorny, J., 2001. Static Pages are Dead: How a Modular Approach is Changing Interaction Design. ACM Interactions 8 (5), 19–24.

Quan, D., Huynh, D., Karger, D., Miller, R., 2003. User Interface Continuations. In: Konstan, J. (Ed.), Proceedings of the Sixteenth Annual ACM Symposium on User Interface Software and Technology (UIST). ACM Press, New York, NY, pp. 145–148.

Queinnec, C., 2000. The Influence of Browsers on Evaluators or, Continuations to Program Web Servers. ACM SIGPLAN Notices 35 (9), 23–33.

Ramakrishnan, N., Capra, R., Pérez-Quiñones, M., 2002. Mixed-Initiative Interaction = Mixed Computation. ACM SIGPLAN Notices 37 (3), 119–130.

Ramakrishnan, N., Perugini, S., 2001. The Partial Evaluation Approach to Information Personalization. Tech. Rep. cs.IR/0108003, Computing Research Repository (CoRR).

Randell, B., Buxton, J. (Eds.), 1969. Software Engineering Techniques: Report of a Conference sponsored by the NATO Science Committee. Brussels, Scientific Affairs Division, NATO (1970), Rome, Italy, 164 pages.

Reps, T., Rosay, G., 1995. Precise Interprocedural Chopping. ACM SIGSOFT Software Engineering Notes 20 (4), 41–52.

Reps, T., Turnidge, T., 1996. Program specialization via Program Slicing. In: Danvy, O., Glück, R., Thiemann, P. (Eds.), Proceedings of the Dagstuhl Seminar on Partial Evaluation; Lecture Notes in Computer Science. Vol. 1110. Springer-Verlag, Schloss Dagstuhl, Wadern, Germany, pp. 409–429.

Ricca, F., Tonella, P., 2001a. Understanding and Restructuring Web Sites with ReWeb. IEEE MultiMedia 8 (2), 40–51.

Ricca, F., Tonella, P., 2001b. Web Application Slicing. In: Proceedings of the International Conference on Software Maintenance (ICSM). IEEE Computer Society, Los Alamitos, CA, pp. 148–157.

Ricca, F., Tonella, P., Baxter, I., 2001. Restructuring Web Applications via Transformation Rules. In: Proceedings of the First International Workshop on Source Code Analysis and Manipulation (SCAM). IEEE Computer Society, Los Alamitos, CA, pp. 150–160.

Ricca, F., Tonella, P., Baxter, I., 2002. Web Application Transformations based on Rewrite Rules. Information and Software Technology 44 (13), 811–825.

Sacco, G., 2000. Dynamic Taxonomies: A Model for Large Information Bases. IEEE Transactions on Knowledge and Data Engineering 12 (3), 468–479.

Sacco, G., Tzitzikas, Y. (Eds.), 2009. Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience. Springer, Berlin.

Spiliopoulou, M., 2000. Web Usage Mining for Web Site Evaluation. Communications of the ACM 43 (8), 127–134.

Srivastava, J., Cooley, R., Deshpande, M., Tan, P.-N., 2000. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. ACM SIGKDD Explorations 1 (2), 12–23.

Takano, A., 1991. Generalized Partial Computation for a Lazy Functional Language. ACM SIGPLAN Notices 26 (9), 1–11.

Taylor, A. (Ed.), 2000. Waynar's Introduction to Cataloging and Classification, Ninth Edition. Libraries Unlimited, Inc., Englewood, CO.

Tip, F., 1995. A Survey of Program Slicing Techniques. Journal of Programming Languages 3 (3), 121–189.

Ullman, J., 1997. Elements of ML Programming, Second Edition. Prentice Hall, Upper Saddle River, NJ.

Venkatesh, V., 1991. The Semantic Approach to Program Slicing. ACM SIGPLAN Notices 26 (6), 107–119.

Wegman, W., Zadeck, F., 1991. Constant Propagation with Conditional Branches. ACM Transactions on Programming Languages and Systems 13 (2), 181–210.

Weiser, M., 1979. Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method. Ph.D. dissertation, University of Michigan.

Weiser, M., 1982. Programmers use Slices when Debugging. Communications of the ACM 25 (7), 446–552.

Weiser, M., 1984. Program Slicing. IEEE Transactions on Software Engineering 10 (4), 352–357.

Williams, M., 1984. What makes RABBIT run? International Journal of Man-Machine Studies 21 (4), 333–352.

Wilson, R., Bergeron, R., 1999. Dynamic Hierarchy Specification and Visualization. In: Proceedings of the IEEE Symposium on Information Visualization (INFOVIS). IEEE Computer Society, Los Alamitos, CA, pp. 65–72.

Yankelovich, N., 1996. How Do Users Know What To Say? ACM Interactions 3 (6), 32–43.