Recognizing the Plans of Screen Reader Users

Marcus J. Huber
Intelligent Reasoning Systems
4976 Lassen Drive
Oceanside, California, 92506
marcush@marcush.net

Richard Simpson
Dept of Rehabilitation Science and Technology
University of Pittsburgh
Forbes Tower, Suite 5044
Pittsburgh, PA 15260
ris20@pitt.edu

Abstract

Less than half of the individuals of working age with visual impairments are employed, and a significant barrier to employment is effective computer access. Some of the screen reader applications offer some help but have limited context sensitivity and are of limited use in applications with dynamic "interfaces" like web pages. Sophisticated screen readers provide aid through application-specific scripts but their full potential is reduced by a users' limited awareness of the scripts and the difficulty in programming and modifying scripts. Technologies such as plan recognition and the automation and optimization of script generation that provide a more adaptive interface for the user will significantly improve computer accessibility to the visually impaired. In this paper, we discuss the addition of probabilistic plan recognition capabilities and supporting framework to an industry leading screen reader to improve accessibility of computers to the visually impaired at work and at home. We discuss both the underlying model-based probabilistic procedural model as well as the system developed.

Introduction

Effective access to computers is becoming increasingly crucial for academic and vocational success. It is predicted that 60 percent of U.S. jobs will require computer skills within the next five years (U.S. Department of Education 1996). Currently, less than half of the individuals of working age with visual impairments are employed (McNeil 1993), and a significant barrier to employment is effective computer access. In particular, manipulating information on the WWW, our application focus, is rapidly becoming a crucial computer skill.

Screen readers, applications that audibilize the text on computer screens, provide some support. For example, a visually impaired user must hit the downarrow or tab keys to move to a subsequent line of text, which is audibilized (read aloud to the user via text to speech technology) as the user performs the navigation. When web browsing, another keystroke moves from link to link, audibilizing the link contents (text and URL). However, manually finding relevant information and links on web pages can involve dozens, if not hundreds, of manual navigation actions, is very difficult to repeat, and takes substantial time to wait for text to be read.

Scripts can automate such navigation tasks, but the few screen readers that support scripts are still limited in a number of ways. For example, users need to learn that scripts exist before they can invoke them (usually with a key combination that's bound to a script), and there are different scripts associated with different applications (e.g., one set of scripts for a particular word processor, another set of scripts for a different word processor (with some scripts appliable to both), another set of scripts for a particular spreadsheet, ...). Knowing which scripts are available and how to invoke them presents a steep learning curve to new and even intermediate users.

Often the user would be best suited with a fully or partially customized script. Writing scripts is typically a very difficult prospect for most users, and any development that makes generating or customizing scripts can greatly benefit a user. Screen readers at the time our project started did not provide any mechanism for users to easily create new scripts nor to easily modify them, relying upon standard editing and debugging paradigms that aren't tailored to the visually impaired.

Futhermore, a fundamental flaw in all screen readers is that they generate their output based strictly upon an application's visual display without any regard to the current context of the application (Raman 1997). Given that browser applications change little or not at all while the underlying data – the web pages that are the *real* focus of the user – can change redically from web page to web page, the current screen reader functionality and scripts provide little more than a starting point to the user. Failing to appreciate the state of an application and the user's goals causes screen readers to 1) provide unnecessary information, 2) overlook relevant connections between output in different parts of the display, and 3) fail to emphasize especially important data.

The research performed and the software developed during this project has been focused on a sophisticated screen reader with scripting capability called JAWS [1]. First, JAWS has been extended to simplify and speed script generation. This has been done by developing a macro recorder, developing algorithms to reason about the *goal* of the set of macro-recorded actions, and developing algorithms that rea-

[1]Freedom Scientific Blind/Low Vision Group, 11800 31st Court North, St. Petersburg, Florida 33716.

son about how to produce a script that is much shorter and more efficient than that produced using a typical simple macro recorder scheme. In addition, we have developed the infrastructure required to recognize existing scripts to identify when the user is manually performing a task for which a script is available. Performing this detection and subsequently notifying the user of this fact has the potential to save the user tremendous time. This same infrastructure also supports using existing scripts as the basis for recognizing scripts *similar* to the user's particular needs that can be used with minimal tailoring, further simplifying and speeding script generation. In summary, what this means to the end user is 1) that they will be made aware of the existance of the scripts that they are performing manually and instructed on how to execute the matching script, 2) that they will be allowed to make errors during script generation and web page navigation and still be able to constructively perform those tasks, 3) that they will find it much easier to generate new scripts that are compact and efficient, and 4) that they will be able tailor existing scripts significantly easier.

Enhancing A Screen Reader: Overview

One capability that provides a significant improvement in screen reader technologies is the ability to identify the user's intentions as the user is performing a task. This capability, called *plan recognition*, can provides assistance to the user as well as to provide useful contextual information to our script generation and optimization extensions discussed below. Our plan recognition mechanism is based on the ASPRN (Automated Synthesis of Plan Recognition Networks) system's probabilistic modeling theories and implementation (Huber 1996; Huber, Durfee, & Wellman 1994). ASPRN takes procedures as its input, in this case JAWS scripts, and produces specially constructed probabilistic models that we call Plan Recognition Networks (PRNs, a particular instantiation of belief networks) that models those procedures. Because PRN computations are based on probabilistic models and relationships, belief networks are well suited for dealing with uncertain, conflicting, or extraneous information, something often encountered in user interfaces. Other plan recognition algorithms using probabilistic representations, for example ((Kaminka, Pynadath, & Tambe 2002; Paek & Horvitz 2000; Goldman, Geib, & Miller 1999)), might be applicable to screen readers, but only ASPRN is based on the directly executable plan representations that are crucial to this domain. Furthermore, users will almost certainly deviate from standard task templates whether it be intentionally or unintentionally, something non-probabilistic representation schemes (e.g., (Broverman, Huff, & Lesser 1987; Vilain 1990; Goodman & Litman 1990)) cannot deal with in a natural or pragmatic way.

Our system's design is shown in Figure 1. All of the components illustrated in Figure 1 have been implemented and are operational. The JAWS screen reader is central to all of our current work, providing a good basis of existing functionality and internal representations for improving computer access to the visually impaired.

The Script Library maintains all of JAWS' application-specific scripts, each of which accomplishes a small, specific, task. The commercially distributed version of JAWS comes with a large corpus of scripts. These scripts are useful only for using applications like Internet Explorer in ways that are not content specific. The pre-existing JAWS scripts were therefore of limited use for web access. Early feedback from visually-impaired clients of The Lighthouse of Houston[2] suggested that we initially concentrate on several tasks that the visually impaired frequently perform with web browsers (e.g., retrieving a weather forecast, online purchasing, checking stock ticker values) rather than anything job-specific. With this in mind, we developed scripts for performing each of these tasks and also a number of subscripts that perform common rudimentary functions. The latter of these, the low-level subscripts, ultimately play an important role in distinguishing between scripts when multiple strategies can be used interchangeably within a single high-level task and when multiple high-level tasks are differentiated based on which strategies are used.

Before a user interacts with the system, we present the scripts in the Script Library to ASPRN for convertion into PRNs. The resulting PRN files are collected and placed in what is called the PRN Library.

The Script Generation Interface (SGI) takes a macro-recorded sequence of user actions (from the Macro Recorder) and creates an optimized script which is then added to the Script library and passed through ASPRN to create a PRN to add to the PRN Library. In addition to script optimization, the SGI presents an interface[3] to the user so that the pre- and post-optimized script can be reviewed and modified by the user in a number of ways before being accepted or rejected.

The Plan Recognition Engine (PRE) accesses the PRNs in the PRN Library and uses these probabilistic models in combination with information about the user actions and context to determine whether and how best to assist the user. At the current time, the PRE is used primarily by the SGI during macro-recorded script optimization.

Constructing Belief Networks

Structure

The execution flow that ASPRN follows is shown in Figure 2. ASPRN first parses a JAWS script file, possibly containing many JAWS scripts and function definitions (the former possibly invocable by users, the latter only accessible at the programming level), into a generic internal procedural model. This internal object-oriented representation provides a reusable framework into which to parse procedural constructs ubiquitous to programming languages:

- Simple sequences of actions
- Hierarchical invocation (e.g., subfunctions, subgoals)
- Conditional execution (e.g., IF-THEN-ELIF-ELSE in programming and scripting languages like JAWS, AND and OR constructs in AI/agent languages)

[2]The Lighthouse of Houston, 3602 West Dallas, Houston, Texas, 77019.

[3]Tested and evaluated to make sure it is accessible to the visually impaired

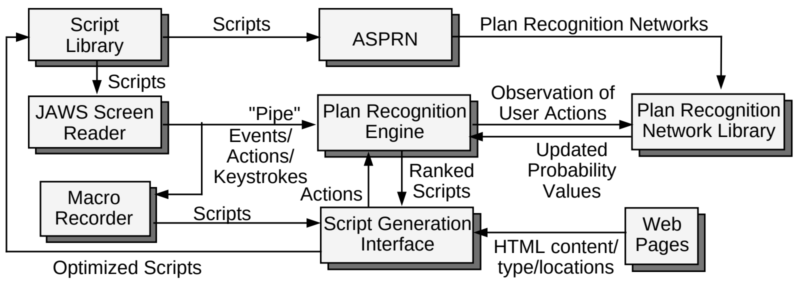

Script Library
Scripts
ASPRN
Plan Recognition Networks
Scripts
JAWS Screen Reader
"Pipe" Events/ Actions/ Keystrokes
Plan Recognition Engine
Observation of User Actions
Plan Recognition Network Library
Updated Probability Values
Macro Recorder
Actions
Ranked Scripts
Scripts
Script Generation Interface
Web Pages
HTML content/ type/locations
Optimized Scripts

Figure 1: Architectural diagram of our enhanced screen reader system.

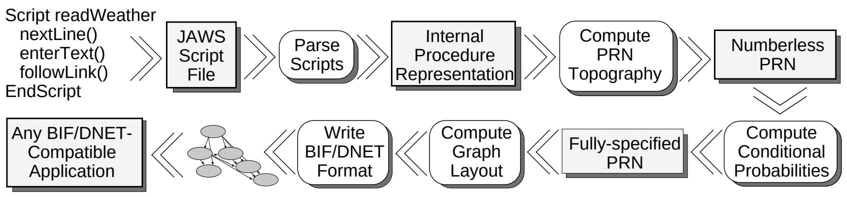


Figure 2: The ASPRN process of taking a JAWS script and constructing a specially-designed belief network called a Plan Recognition Network (PRN).

- Iteration (e.g., WHILE and DO loops)

For simple, abstract examples of JAWS instances of the constructs listed above, see Figure 4(A), (B), (C), and (D), respectively. The JAWS scripting language is a complex, fully featured programming language, including all of the above programming constructs above as well as features such as declarations and use of constants and variables, parameters to function, value returns from functions, events, and nested file inclusion.[4]

ASPRN computes a PRN topography based on the syntactic nature of the procedures parsed (see below). After this, the conditional probability values associated with the PRN instance are computed based on a domain-independent procedural model. The procedural constructs in JAWS scripts can be instantiated into PRNs for any possible combination of chaining, nesting, and hierarchical invocation (except recursion) and results in a coherent probability distribution over the entire network.

For debugging and analysis purposes, ASPRN next determines a visual layout for the PRN for when it is loaded into belief network visualization applications. Finally, ASPRN writes out the completed PRN into a file (in either BIF or DNET formats - standard formats for belief network files) for adding to the PRN Library and subsequent use by the PRE.

The basic modeling theories, algorithms, and key application concepts from the original ASPRN system (Huber 1996; Huber, Durfee, & Wellman 1994) were modified in a couple places to suit the specifics of the JAWS scripting language. For example, the agent plan languages modeled to date allow for multiple possible plans for each subgoal invocation, unlike JAWS scripts (and most high-level programming languages), in which there is a unique function or macro definition for each call in a script. This resulted in much simpler PRNs due to the removal of the necessity to model the one-to-many relationship. Conditional execution semantics also differed enough between agent languages and JAWS scripts to warrant a significant modification. In this case, agent plans contained AND/OR constructs that are not mutually exclusive as they are in standard IF-THEN-ELSE styles of conditional execution. In this situation too, the resulting PRNs are simpler than those for agent-based languages.

More details of how ASPRN creates belief networks for the various JAWS script procedural constructs are shown in Figure 3 and Figure 4. Note again that during PRN construction, ASPRN composes the individual belief network pieces built for each script construct into a single fully-connected belief network. In Figure 4, ovals represent random variables and arcs represent probabilistic relationships between variables. Belief network nodes are created in a regular, systematic way. Each node is associated with a certain semantic type by ASPRN, according to the JAWS script element that it is modeling. Four semantic types are currently modeled: each *script* node represents a script, whether that particular script is being executed; each *action* node represents a script action/function, i.e., whether that action has been performed; an *evidence* node is associated with each action node to model the fidelity of the system's ability to correctly sense when the action has been performed (typically a domain specific relationship); and each *condition* nodes represent the boolean expressions of IF-ELIF conditions and WHILE loops.

Connections between PRN nodes are created in a regular, systematic way. ASPRN maintains a semantic model of the arcs between nodes, remembering the reason the link was

[4]Describing all of the details of the JAWS scripting language is beyond the scope of this paper, but can be found at www.freedomscientific.com/fs_support/doc_scriptfunction.asp.

```
Procedure Construct_PRNs {
  for each top_level_script {
    create_procedure_node top_level_script_name
    Map_Procedure top_level_script
  }
  Calculate_Probabilities
}

Procedure Map_Procedure {
  for each construct in procedure {
    Map_Construct construct
    link_constructs prior_construct construct
  }
}

Procedure Map_Construct {Sequence} {
  for each construct in sequence {
    Map_Construct construct
    link_constructs prior_construct construct
  }
}

Procedure Map_Construct (IF-ELIF-THEN-ELSE) {
  for each branch in construct {
    create_condition_node branch_condition_expression
    for each construct in the branch {
      Map_Construct construct
      link_constructs prior_construct construct
      link_constructs construct condition_node
    }
  }
}

Procedure Map_Construct (Iteration) {
  create_condition_node loop_continuation_expression
  for each construct in loop body {
    Map_Construct construct
    link_constructs prior_construct construct
    link_constructs construct condition_node
  }
  link_constructs construct construct_after_loop
  link_constructs construct_after_loop condition_node
}
```

Figure 3: Pseudocode for ASPRNs key modeling algorithms.

created between a parent node and a child node. The arc semantics created by ASPRN for JAWS scripts are *parent script* arcs that originate at a script node and terminate at all nodes directly related to the script, *prior action* links that originate at an action node and terminate in a node for the next executable action(s), *evidence* links that originate at an action node and terminate at its evidence node, and *condition* links that originate at action nodes that provide evidence for their truth value and terminate at a condition node.

In all cases, a script node is connected to each of its component actions with a link. A link is also created between action nodes if the actions they represent are consecutive in a script, with the arc going from the prior action to the posterior action, to model the temporal order of execution. Figure 4(A) illustrates this, showing how a sequence of actions in a script is converted into a linearly connected construction of random variables representing the order of the actions in the sequence. As mentioned above, each script primitive action also has an associated evidence node that is used to model the reliability of observations and is typically where observations are ascribed to during plan recognition. Our modeling of temporal relationships between actions may be changed to a 2-step or 3-step temporal model in the future if it is deemed advantageous.

In Figure 4(B), we illustrate how iteration is modeled. In this case, the action sequence within the loop is modeled in isolation. ASPRN then constructs a condition node representing the loop continuation expression. All of the actions inside the WHILE loop consitute evidence that the loop expression is true, so an arc is added from all of the actions within the loop to this condition node. Furthermore, the fi rst action just past the loop reflects negatively upon the truth value of the loop condition (as do the subsequent actions, but ASPRN does not model this) and an arc is added between the node for this action and the condition node. The evidence nodes in this and subsequent models are not shown to save space.

Figure 4(C) illustrates how JAWS' conditional branching is modeled. First, the action sequences for each branch (i.e., the IF branch, any ELIF branch, any ELSE branch) is constructed in isolation, with the only links created at this point being to each action node from the parent script and any prior actions. Because there is a condition expression associated with each branch, except for the fi nal ELSE action sequence, a new condition node is created to represent each of these expressions. A link is then created from each action in a branch to its respective condition node, reflecting the truth value of the expression.

JAWS scripts are often nested, with scripts invoking functions and other scripts. Figure 4(D) illustrates how a multilevel script is modeled. In such a case, the script of function being invoked is fi rst treated like any other action, with links created to it from the nodes representing the invoking script, any prior action, etc., and from it to any condition nodes and subsequent actions. Then the script node for the invoked script/function is treated as if it were a top-level script node and its own script is then constructed, recursively.

Function and script arguments are not modeled within the belief network topology or probabilities explicitly, but are maintained as part of the description of each belief network node for subsequent use when adding user observations as evidence during the plan recognition process. This represents a signifant improvement over the ASPRN described in (Huber, Durfee, & Wellman 1994; Huber & Durfee 1995; Huber 1996) as it allows for much better fidelity in matching observations with PRN action nodes.

Probabilities

To specify the probabilities for each PRN, we identified all classes of combinations of arcs for each node type that might arise and specified values for each state value for each type of variable (GOAL, ACTION, CONDITION, and EVIDENCE). Through careful selection, some trial and error, and quite a bit of experience applying PRNs to problems, we have determined a generic set of prior and conditional probabilities that work well across multiple domains (Huber & Hadley 1997; Huber & Simpson 2003). The pattern–based approach to probability determination that we constructed requires a fairly large number of patterns (32). However, this mechanism significantly reduces the total number of manual probability assessments required to create fully specified PRNs, which is its primary purpose. It should be noted that the probabilities encoded do not represent actual statistical relationships between domain actions and scripts (e.g., top-level scripts are given equivalent priors), but the structural and execution/observation relationships between script components. As such, the posteriors produced by the PRNs do not represent true statistical posteriors (e.g., 0.10 does not represent a 10% chance of occurrence) but can be evaluated for relative change and comparing values between nodes.

An ASPRN node's conditional probability table is determined by the combination of the node's own semantics and the semantics of the arcs incoming to the node. For example, an action node with an arc from a parent script node and a prior action node will have a different probability table than an action node with an arc from a parent script node but no prior action. It will also be different from an action node with an arc from a parent script node and a prior action that's actually a script node. These patterns, called conditioning cases, are used to specify a probability value for any variables given a particular set of incoming arcs with specified parent state values. The conditioning case patterns represented within ASPRN are an exhaustive list of possible combinations. These entries coincide with the possible arc semantics classifications, one entry for each of the four arc semantic classifications listed above. Each entry α is a pair, (Ω, Π), where $\Omega \in \{$ *DONTCARE, NONE, ONE, ONEORNONE, ONEORMORE, ALL*$\}$ is a constraint specification, and $\Pi \in \{$ *Inactive*, *Active*, *Achieved* $\} \parallel \{$ *NotPerformed*, *Performed* $\} \parallel \{$ *False*, *True* $\}$ is a list of parent state values that must satisfy the entry's constraint specification (for script, action, and condition nodes respectively). The semantics associated with each possible constraint specification is given below.

- *DONTCARE* – Ignore any incoming arcs with this entry's semantic classification. I.e., the node is independent with respect to arcs with this semantics.
- *NONE* – No arcs with this entry's semantic classification may have a parent's state value specified within Π
- *ONE* – Only one of the arcs with this entry's semantic classification may have a parent's state value specified within Π
- *ONEORNONE* – Either no arc, or only one arc with this entry's semantic classification may have a parent's state value specified within Π
- *ONEORMORE* – At least one of the arcs with this entry's semantic classification must have a parent's state value specified within Π
- *ALL* – All of the arcs with this entry's semantic classification must have a parent's state value specified within Π

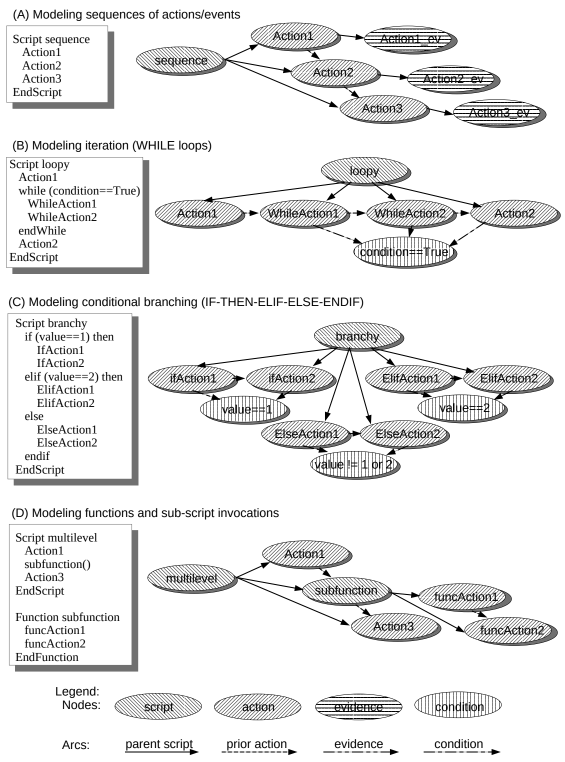
Table 1 shows an example of a fully specified pattern. The pattern shown in this figure matches the cases for an action (not the first action) in a simple sequence of actions where the parent goal has the state *Active* and the action immediately prior to the variable for which this probability is being specified is either *Inactive* (it could be a *script* node) or *NotPerformed* (it could be an *action* node). All incoming arcs with other semantics are ignored. This probability pattern does *not* apply to the first action of a simple sequence due to the constraint specification that *one* (not zero, not more than one) incoming arc must be from a variable representing the action immediately preceding the action for which the probability is being specified.

Associated with each conditioning case pattern is a set of probabilities to assign depending upon the state value of the variable for which the conditional probabilities are being determined. In the case of a *script* node, $\alpha(\Omega, \Pi)$ = { 0.95, 0.04, 0.01 } corresponding to the state values of $\{$*Inactive*, *Active*, *Achieved*$\}$ respectively, while for an ACTION node, $\alpha(\Omega, \Pi)$ = { 0.95, 0.05 }, which would correspond to state values of $\{$*Notperformed*, *Performed*$\}$. All conditional probability table entries for all variables in the plan recognition network that have a conditioning case that meets this criterion have their values set according to this particular set of values. In short, PRNs start with a generic case-based model of the syntactic, temporal, and causal relationships between belief network nodes, not the actual in-use statistical probabilities (which can eventually be learned).

Runtime Operation

In this section, we briefly describe how our enhanced JAWS system works. All of the components illustrated in Figure 1 have been implemented and are operational.

The mode of assistance fully implemented at this point is during the generation of scripts. In this mode, the user invokes the Macro Recorder to start recording actions (possibly containing mistakes of various types), performs the actions required to achieve a particular task, and then turns the Macro Recorder off. The SGI is then invoked, which uses the PRE to analyze the sequence of actions and events in order to optimize the raw recorded command and event se-



```
Script sequence
  Action1
  Action2
  Action3
EndScript
```

```
Script loopy
  Action1
  while (condition==True)
    WhileAction1
    WhileAction2
  endWhile
  Action2
EndScript
```

```
Script branchy
  if (value==1) then
     IfAction1
     IfAction2
  elif (value==2) then
     ElifAction1
     ElifAction2
  else
     ElseAction1
     ElseAction2
  endif
EndScript
```

```
Script multilevel
  Action1
  subfunction()
  Action3
EndScript

Function subfunction
  funcAction1
  funcAction2
EndFunction
```

Figure 4: PRN constructs that model various JAWS script constructs.

Ω	Π	Arc Semantics Classification
ONE	Active	Parent Script
ALL	Inactive, NotPerformed	Prior Action
DONTCARE	ANY	Prior Branch
DONTCARE	ANY	Loop Condition
DONTCARE	ANY	Negative Loop Conditions

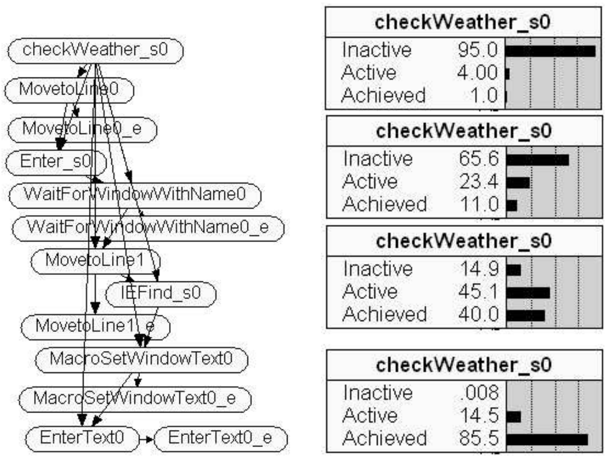
Table 1: An example of an ASPRN probability pattern. This pattern matches all non–first–action actions of a sequence (indicated by a Π value other than *ANY* in the Prior Action row) within a plan where the parent goal is *Active* and the previous action/subgoal in the sequence is either *Inactive* or *NotPerformed*.
There are 31 other such patterns defined that cover all modeled network possibilities.

```
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Tab()
PerformScript Enter()
WaitForWindowWithName("CNN.com")
PerformScript Tab()
PerformScript Tab()
PerformScript IEFind()
TypeKey("z")
TypeKey("i")
TypeKey("p")
TypeKey("Enter")
TypeKey("9")
TypeKey("2")
TypeKey("0")
TypeKey("5")
TypeKey("6")
TypeKey("Enter")
```

Figure 5: The pre-optimized script for reading weather at CNN.com (as recorded by the Macro Recorder).

```
Script checkWeather()
  MovetoLine(39)
  PerformScript Enter()
  WaitForWindowWithName("CNN.com")
  MovetoLine(3)
  PerformScript IEFind()
  MacroSetWindowText("zip")
  EnterText("92056")
EndScript
```

Figure 6: The post-optimized script for reading weather at CNN.com.



checkWeather_s0	
Inactive	95.0
Active	4.00
Achieved	1.0

checkWeather_s0	
Inactive	65.6
Active	23.4
Achieved	11.0

checkWeather_s0	
Inactive	14.9
Active	45.1
Achieved	40.0

checkWeather_s0	
Inactive	.008
Active	14.5
Achieved	85.5

Figure 7: Left, the PRN constructed by ASPRN (evidence nodes not shown). Right, the probability distribution for the script being performed by the user with no observations (i.e., priors) at top, MoveToLine observed (second), Enter observed (third), and all actions observed (bottom). Note the gradual progression from Inactive to Achieved (i.e., completed).

quence. The PRE uses ASPRN's API to make observations[5] at several levels within the PRNs: at evidence nodes when there is uncertainty related to correlating Macro Recorder entries with script actions; at action nodes (one level up from the evidence level) when this can be done with surety; and at the script nodes when this can be directly ascertained by observations[6]. The PRE then uses the ASPRN API to trigger Bayesian updating in the PRNs and to query the PRN for the posteriors of nodes of interest. Based on the values of the posteriors from the PRE, the SGI makes decisions on how best to optimize the script by replacing multiple Macro Recorder raw actions with a single macro or function invocation. The user can review and modify the optimized script in a number of ways before finally accepting or rejecting it. Acceptance or rejection of the replacement suggestions will be used by planned future Bayesian learning capabilities to adjust the internal PRN models to tailor the models to the user.

During what we consider normal use of the extended JAWS system, the user will be navigating between and within web pages while JAWS observes the user's actions and screen events and "pipes" them to the PRE. After each observed action/event, the PRE adds this information as evidence into the PRNs in the PRN Library and invokes belief updating. The PRE sorts the scripts based on posterior probabilities to determine the most likely script being pursued by the user.

As an example, the user recorded the macro shown in Figure 5. This relatively lengthy, overly verbose script was optimzed by the system into the script shown in Figure 6. Figure 7 illustrates how PRN posteriors react to evidence of the user's actions[7]

PRNs modeling scripts that are close but do not quite match the evidence pattern do not react as strongly and result in lower posterior probabilities, allowing the PRE to discriminate between alternatives. Once a script has a sufficiently high likelihood of being pursued (initial user studies have determined roughly appropriate value for this but more need to be performed), the system can inform the user that a script exists that can perform their task much more efficiently or is close but needs to be tailored slightly for the user's current task. The Bayesian learning capabilities that will be added at a later stage will take the user's acceptance or rejection of this suggestion and gradually adjust the PRNs to the user's preferences (the rate of which will also be determined through user studies). This mode of operation is almost fully implemented, lacking user prompting for assistance and final integration of the action/event "pipe" with the PRE.

[5]These are currently ascribed with certainty, but observations to ASPRN can also be made as likelihood values.

[6]Evidence can be made at condition nodes as well but we have not yet needed to do this.

[7]It is beyond the scope of this paper to describe how such a network behaves for all observation patterns (e.g. those with irrelevant or erroneous actions). For more details refer to (Huber, Durfee, & Wellman 1994; Huber 1996).

Future Work

There remains a large number of future research and implementation issues in addition to the possibilities alluded to earlier. For example, we have performed some initial user trials with experienced JAWS users to evaluate the system in whole and in part, many more aspects of the system need to be evaluated in more depth before actually fielding it. For example, a number of variations and specializations of PRN topographies and conditional probabilities need to be explored to determine the optimal configuration for screen reader tasks. There are tradeoffs in modeling accuracy and computational effort that must be explored to find the correct balance. Also, while we have an internal object-oriented procedural model, we have yet to take advantage of recent object-oriented probabilistic modeling research (e.g., (Barry, Laskey, & Brouse 2000; Pfeffer *et al.* 1999). Explicitly modeling typical user deviations/errors (Calistri-Yeh 1991) could also be used to advantage .

With respect to application domains, our work to date with the visually impaired has focused almost exclusively on tasks that are performed on the World Wide Web. Our continuing work will also focus on the vocational and educational uses of the proposed software involving off-line applications. In addition, a formal experiment will be conducted to examine the proposed software's ability to reduce the time and cost associated with (1) making a worksite accessible for the visually impaired and (2) training a visually impaired employee to use an accessible workstation.

Acknowledgements

We would like to acknowledge the National Science Foundation, which funded this project under grant DMI-0091590. We would also like to acknowledge the work of Lin Ma, a graduate student at the University of Pittsburgh, and Liela Garcia, a software engineer at Freedom Scientific's Blind/Low-Vision Group. Both have contributed significantly to the implementation of various aspects of this project.

References

Barry, P.; Laskey, K.; and Brouse, P. 2000. Development of Bayesian Networks from Unified Modeling Language (UML) Artifacts. In *Proceedings of the Twelth Software Engineering/Knowledge Engineering 2000 Conference*.

Broverman, C.; Huff, K.; and Lesser, V. 1987. The Role of Plan Recognition in Design of an Intelligent User Interface. In *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, 863–868.

Calistri-Yeh, R. J. 1991. Classifying and detecting plan-based misconceptions for robust plan recognition. *AI Magazine* 12(3):34–35.

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A New Model of Plan Recognition. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 245–254.

Goodman, B., and Litman, D. 1990. Plan Recognition for Intelligent Interfaces. In *Proceedings of the Sixth Conference on Artificial Intelligence Applications*, 297–303.

Huber, M., and Durfee, E. 1995. Deciding When to Commit to Action During Observation-based Coordination. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, 163–170.

Huber, M. J., and Hadley, T. 1997. Multiple Roles, Multiple Teams, Dynamic Environment: Autonomous Netrek Agents. In Johnson, W. L., ed., *Proceedings of the First International Conference on Autonomous Agents*, 332–339. New York: ACM Press.

Huber, M. J., and Simpson, R. 2003. Plan Recognition to Aid the Visually Impaired. In Brusilovsky; Corbett; and de Rosis., eds., *Proceedings of the Ninth International Conference on User Modeling*, 138–142. New York: Springer.

Huber, M.; Durfee, E.; and Wellman, M. 1994. The Automated Mapping of Plans for Plan Recognition. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 344–351.

Huber, M. 1996. *Plan-Based Plan Recognition Models for the Effective Coordination of Agents Through Observation*. Ph.D. Dissertation, The University of Michigan.

Kaminka, G.; Pynadath, D.; and Tambe, M. 2002. Monitoring teams by overhearing: A multiagent plan-recognition approach. *Journal of AI Research* 17:83–135.

McNeil, J. 1993. Americans with Disabilities: 1991-92. U.S. Bureau of the Census. Current Population Reports, P70-33, U.S. Government Printing Office, Washington, DC.

Paek, T., and Horvitz, E. 2000. Conversation as Action Under Uncertainty. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, 455–464.

Pfeffer, A.; Koller, D.; Milch, B.; and Takusagawa, K. T. 1999. SPOOK: A System for Probabilistic Object-Oriented Knowledge Representation. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 541–550.

Raman, T. 1997. The Audible WWW - The World In My Ears. http://simon.cs.cornell.edu/Info/People/raman/publications/www-access-97/.

U.S. Department of Education. 1996. Getting America's Students Ready for the 21st Century. Washington, DC. http://www.ed.gov/Technology/Plan/NatTechPlan/title.html.

Vilain, M. 1990. Getting Serious About Parsing Plans: a Grammatical Analysis of Plan Recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 190–197.