研究課題別評価

1. 研究課題名：計算状態パーソナル・スクラップブック

2. 研究者氏名：リチャード ポッター

3. 研究の狙い

The goal of this research is to investigate new programming tools that work by saving and restoring intermediate computation state. The same basic idea has brought important benefits to other types of tools and applications. Scientific and other distributed applications can benefit by saving an application in mid-execution and moving it to a machine with more resources. Other applications gain fault tolerance by saving application state for rolling back to a safe checkpoint in case of hardware or software failures. However, few benefits have been explored for programming tools that can save and restore the computation state of a program under development.

One reason to expect benefits is that quick initialization of programs in mid-execution can make it possible to focus programming tools and programmer attention in new creative ways. People who are learning new programming skills can benefit because it allows a wider range of techniques to be applied to a program that has been subdivided into more manageable pieces. An additional general benefit is that program state can be enumerated more efficiently and methodically, which enables model checking on actual implementations. The questions for this research are what tools can make these potential benefits practical for actual programming activity and what infrastructure will make these tools easy to implement.

Tools based on Computation Scrapbooks have potential to be simple because it is simple and increasingly practical to copy computation state. It is conceptually simple because, like photo copying complex information on a sheet of paper, it is not necessary to understand the information to easily copy it. Copying computation state is practical because hard disk sizes and processor speeds are fast enough to copy realistic computation state quickly.

The challenge of this research is to demonstrate that Computation Scrapbooks can be both useful and practical without adding too much complexity to the simple core functionality that copies the computation state. Thus it is important to distinguish between the *core functionality*, which is well defined and clearly practical, and the *extended functionality*, which may be less well defined and only be possible for certain special cases.

The approach of the research has been to create two systems. The first system is for fast prototyping various Computation Scrapbook based tools. It has limited core functionality, but is flexible for exploring the various types of extended functionality necessary to support the tools. The second system is designed for actual use by real users. It has more rigorous core functionality that is general, however the range of extended functionality and tools is less than the first system.

The name "Computation Scrapbook" represents the core infrastructure itself, and is intended to convey that multiple snapshots of computation state are saved and used in creative ways. It encompasses functionality to save, organize, and restore snapshots of computation state. To support programming tools, a Computation Scrapbook must support thread persistence, because use in programming tools requires checkpointing at a fine granularity and therefore the stack states must be preserved. Also, multiple persistent snapshots will be required for several of the programming tools.

4. 研究結果

In this research, two different Computation Scrapbooks have been implemented using different techniques. SBDebug, the first system implemented, captures the state of Lisp programs that run in the Emacs text editor. It creates snapshots that are typically less than 10KB in size. It works by

instrumenting functions so that they make internal stack frames easy to capture in a top-down manner. Saving and restoring state is quick and the Lisp environment makes it easy to manipulate programs and data. Therefore, SBDebug has been useful for quickly prototyping new Computation Scrapbook tools, although its use is limited to a restricted class of Emacs Lisp programs.

SBUML, the second system implemented, captures the state of the Linux operating system, including all file systems, applications, processes, and kernel state. SBUML is early in development, but it will be useful for creating tools for a wide range of programming languages and will support complex, real-world, multithreaded programs. It works by copying the complete low-level image of all changes to memory and disks of User-Mode Linux, a virtual version of Linux. This makes it less flexible than SBDebug for prototyping innovative tools because the Linux state is more heterogeneous and therefore harder to manipulate than Lisp. Also, the snapshots require more computing resources. Raw snapshots start at about 30MB in size, although snapshots can sometimes be compressed to around 100KB. Saving and restoring raw snapshots takes around 5 seconds on a 2.8gHz dual processor workstation.

Several tools based on the core Computation Scrapbook infrastructure were developed to support various programming activities, such as reading code, writing code, debugging, and testing.

For example, when reading a program, a user can sometimes benefit by watching the program execute using debugging or software visualization tools. However, setting up the debugger so that it shows something useful about a particular section of code can require a lot of skill. A Computation Scrapbook can let a skilled programmer prepare and save such a debugger configuration and share it with other users.

This idea was explored in SBDebug by creating a *snapshot documentation tool* with two features. The first feature lets the experienced programmer paste a hyperlink into source code text. The second quickly takes anybody who selects the hyperlink back to the same debugger configuration. Experimenting with these features has made other uses obvious. For example, hyperlinks could be put in on-line programming language documentation to quickly take readers to live examples of specific language features in action.

While developing a program, sometimes testing has to be delayed until a whole module is completed. Since a snapshot can initialize any part of a larger program, arbitrarily small and partially completed code segments can be tested earlier.

This idea was implemented in SBDebug as a *snapshot test case tool*. To use it, the user sets up the initial computation state for the beginning of the code segment using whatever techniques are most convenient, which might be done by running the program manually, writing a driver, manually editing the state, or some combination of these. This first snapshot is then saved. The user then runs the program to the end of the code segment and saves a second snapshot, possibly editing the computation state manually if the code does not compute the correct outcome. Finally, the user uses the test case tool to create a test case from the two test cases just saved. A second test case tool feature runs the test case. It initializes the code with the first snapshot, runs until the program counter matches that of the target snapshot, and then compares the computation state with the second snapshot to judge if the test passed. A third feature can run a set of test cases with a single command.

Experience with the tool shows that it can be very easy to create a test case in the middle of writing and debugging a program. If later the code is changed, it is easy to rerun the collected set of test cases to quickly check for any new bugs might have been unintentionally introduced.

When writing code, it can sometimes be desirable to give a specific example of what the program does rather than write the abstract code. Although automatic programming techniques are well researched, they only work for very small programs of limited use. Computation Scrapbooks make it possible to apply such techniques to small parts of larger programs, so that they can generate useful code. This can be practical because the snapshot test cases work for code segments that are so short that all possibilities can be enumerated.

This idea can be demonstrated in SBDebug with its *programming by demonstration (PBD) tool*. The user first selects an incomplete or incorrect Lisp expression and also selects a set of test cases. The PBD tool can then enumerate possible expressions until one passes all test cases. In order to make the interface this simple, some plausible defaults were chosen for how to enumerate the expressions. For example, only functions, constants, and other tokens that already appear in the function that contains the original expression are used.
So far this tool is only a proof of concept implementation to show automatic programming is possible for realistic programs when a Computation Scrapbook is used to focus the technique on a small parts of the programs. Some refinements were implemented to delay the inevitable combinatorial explosion. In addition to static type checking, the PBD tool also keeps track of runtime errors to reduce the number of expressions that must be generated and tested.
When verifying the correctness of concurrent programs, model checking techniques can be useful. However, model checking must usually be performed on a separate abstraction that may not match the actual implementation. Computation Scrapbooks can restore the state of an actual implementation repeatedly so that all possible successor states can be generated for model checking. This idea has been demonstrated by enumerating all the possible states of a C implementation of the Dining Philosopher 's algorithm running in SBUML.

5. 自己評価

Overall, the research proceeded steadily with progress in both implementations and high-level ideas. At the beginning of the research period, the high-level ideas were loosely strung together notions about end-user programming, gentle-slope systems, invisible computation state, the increasing practicality of copying computation state, and the idea that interaction with computation state can lead to higher-level understanding. This was enough of a framework to guide the quick implementation of SBDebug. Experience with SBDebug resulted in a clearer understanding of the high-level benefits of Computation Scrapbooks, which mostly come from the ability to quickly initialize programs in mid-execution. From this core benefit, other benefits are easy to explain, including how programs in mid-execution provide context that is useful for gentle-slope systems. Some of the refined high-level ideas were published in a paper that explained how Snapshot Documentation is an appropriate technique for gentle-slope systems. Experience with SBDebug also gave the confidence to start investigating the practicality of Computation Scrapbooks for other systems.
After investigating the practicality of a Computation Scrapbook system for Java, it became apparent that one for Linux might be easier to implement. At first it seemed that checkpointing Linux might be slow and only good for very slow proof-of-concept demonstrations. However, SBUML has turned out to be much faster and useful than expected. It can save and restore snapshots in a few seconds and compress snapshots down to sizes that are practical to download quickly over the Internet.
This research provides a foundation for several useful and challenging research directions. For the short term, most practical benefit will be using SBUML for its Snapshot Documentation potential. SBUML is currently being distributed publicly and has great potential to be useful to researchers and other users. For advanced computer science research, the model checking applications of SBUML show great potential. For ground breaking research, it will be interesting to transfer the extended functionality demonstrated in SBDebug to SBUML and show how the testing and automatic programming tools can work for popular languages like C and Java.

6. 研究総括の見解

ポッター研究者は、プログラムの実行状態の表示、保存、復帰、途中状態からの実行再開などを行うツールがプログラムの理解や開発に有効であるという考えにもとづき、コンピュテーションスクラップブックというツールの研究を行った。この原理自身は特に新しいものではないが、構築されたツールは非常に優れ、有効なものである。ポッター研究者のプログラミングの能力とセンスの良さ、ツール設計

の適切さによるものである。開発された2つのツールのうち、Linuxオペレーティングシステム上のプログラム開発を対象にしたSBUMLは特にそのアイディアの斬新さや適用範囲の広さから期待の大きいツールである。ネットワークプログラミングやシステムプログラミングにとって非常に強力なツールになり得ると考える。

7. 主な論文等

論文

1. O. Sato, R. Potter. M. Yamamoto, and M. Hagiya, UML スクラップブックとスナップショットプログラミング環境の実現, Linux Conference 2003

2. R. Potter and Y. Harada, Additional Context for Gentle-Slope Systems, IEEE Symposia on Human-Centric Computing Languages and Environments (HCC 2003)

3. R. Potter and M. Hagiya, Computation Scrapbooks for Software Evolution, Fifth International Workshop on Principles of Software Evolution (IWPSE 2002)

4. R. Potter, Computation Scrapbooks of Emacs Lisp Runtime State, 第43回プログラミング・シンポジウム

5. R. Potter, Computation Scrapbooks of Emacs Lisp Runtime State, IEEE Symposia on Human-Centric Computing Languages and Environments (HCC 2001)

6. R. Potter, Computation Scrapbooks, 第4回プログラミングおよび応用システムに関するワークショップ (SPA2001)

ソフトウェア

1. SBDebug: A Computation Scrapbook for Emacs Lisp. (Approximately 8500 lines of Lisp)

2. SBUML: A Computation Scrapbook for User-Mode Linux. (Approximately 5500 lines of C and 1800 lines of Shell Script)