Abstract. We report on a dynamic style of proof presentation, which we call proof animation. Our animations are Web-based and build upon the notion of proof plans. Currently our animations are hand-crafted and based upon a single proof plan. However, a proposal is outlined for automating the construction of proof animations for arbitrary proof plans.

1 Introduction

In our experience, one of the most difficult topics for computer science students to understand and master is formal proof. We believe that the static nature of textbook style proofs contributes to this difficulty. Proofs are typically presented as a series of formulae, interleaved with inference rules, i.e. low-level “snap shots” of the reasoning. The onus is on the student to observe the high-level pattern associated within such a low-level presentation. The theorem proving technique known as proof planning [4], can be used to communicate high-level patterns of reasoning. Within proof planning, high-level proof outlines, known as proof plans, are used to both guide the search for proofs and provide proof explanations. A proof plan can be viewed as providing a series of high-level “snap shots” of the reasoning. Explanations generated from proof planners, however, are still essentially static. We argue that a dynamic presentation of a proof will better support the communication of high-level patterns of reasoning. Our goal is to achieve a dynamic style of presentation through Web-based animation: what we call proof animation. The idea is to provide an animation component to proof plans. So far we have only considered the proof plan known as rippling. Our longer term goal, however, is to develop proof animations directly from arbitrary proof plans.

Background on rippling is presented in section §2 through a series of ripple proofs. These proofs are revisited in §3, where the use of Flash for developing animations is explored. A proposal for automating the process of generating proof animations is outlined in §4, while our conclusions are presented in §6.

2 Rippling

Achieving a match between a goal, or part of a goal, and a given hypothesis is a common theorem proving strategy. This strategy has been implemented in a proof plan called rippling. Rippling involves two high-level phases:
– identifying a mismatch between a goal and a given hypothesis;
– reducing the mismatch to so that a match between goal and given can be achieved.

To illustrate, consider a hypothesis of the form:

\[(x + b) + c = x + (b + c)\] (1)

and a goal of the form:

\[(s(x) + b) + c = s(x) + (b + c)\] (2)

Note the similarity between the two formulae, i.e. a copy of (1) is embedded within the goal, i.e.

\[(\ldots x \ldots + b) + c = \ldots x \ldots + (b + c)\]

This embedded formula is known as the skeleton. The goal of rippling is to preserve the skeleton while making progress towards achieving a match between the goal and the given. The terms that cause the mismatch are are delimited by meta-level annotations called wave-fronts. Using shading to represent wave-fronts, then the annotated version of (2) takes the form:

\[
\begin{align*}
\text{(s} & x \text{)} + b + c = \text{s} (x) + (b + c) \\
\end{align*}
\] (3)

Note that the wave-fronts are directional\(^1\), where arrows are used to record the direction in which the wave-front is being moved relative to the skeleton. The process of annotation is automatic, and is also used in annotating rewrite rules. These annotated rewrite rules are known as wave-rules, and are used to reduce the mismatch between goal and given. Note that the application of wave-rules preserves the skeleton and is guaranteed to terminate. Recursive definitions and properties provide a rich source of wave-rules. For example, consider the following recursive definition of +:

\[
\begin{align*}
0 + Y &= Y \\
s(X) + Y &= s(X + Y) \\
\end{align*}
\] (4)

Equation (4) gives rise to the following wave-rules\(^2\):

\[
\begin{align*}
\text{s} (x) + Y &\Rightarrow \text{s} (x + Y) \\
\text{s} (X) + Y &\Rightarrow s(X + Y) \\
\text{s} (X) + Y &\Rightarrow s(X + Y) \\
\end{align*}
\] (5)

\(^1\) Wave-fronts can be directed upward or downward.
\(^2\) We use \(\Rightarrow\) to denote rewrite rules and \(\rightarrow\) to denote logical implication.
Note that wave-rule application requires both object- and meta-level terms to match. A formal account of the ripple proof plan can be found in [3, 7, 5]. Here we concentrate on three example ripple proofs, these proofs provide a basis for comparing a textual style of presentation with our proof animations.

2.1 Rippling Outwards

The most basic form of rippling involves achieving a match by moving wave-fronts outwards through the skeleton. Consider again the ripple problem introduced above:

Given: \((x + b) + c = x + (b + c)\)  \hspace{1cm} (6)

Goal: \(s(x + b) + c = s(x) + (b + c)\)  \hspace{1cm} (7)

Using wave-rule (5) twice, goal (7) ripples on the left-hand side to give:

\[ s((x + b) + c) \Rightarrow s(x) + (b + c) \]  \hspace{1cm} (8)

Further rippling is possible on the right-hand side of (8), again using wave-rule (5), to give:

\[ s((x + b) + c) \Rightarrow s(x + (b + c)) \]  \hspace{1cm} (9)

Note that a partial match between goal and given has been achieved, allowing (9) to be rewritten using (6). Either side of the goal can be rewritten, rewriting the left-hand side gives:

\[ s(x + (b + c)) = s(x + (b + c)) \]

which is trivially true.

2.2 Rippling Sideways & Inwards

We now consider the sideways and inwards variants of rippling. Consider a target hypothesis of the form:

\[ \forall k, l \in \text{list}(\tau). \ qrev(t, k <> l) = qrev(t, k) <> l \]  \hspace{1cm} (10)
and an associated goal of the form:

$$qrev(h :: t, k <> l) = qrev(h :: t, k) <> l.$$  \hspace{1cm} (11)

Note that \textit{qrev} denotes a tail recursive list reversal function, while \textit{<>} denotes list concatenation. The goal of rippling sideways and inwards is to exploit universally quantified variables within a hypothesis. Note that within hypothesis (10), \textit{k} and \textit{l} are universally quantified. The positions within a goal that correspond to such universal variables are delimited using \textit{[. . .]} annotations, which are known as sinks. The ripple problem corresponding to (10) and (11) takes the form:

Given: \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} qrev(t, K <> L) = qrev(t, K) <> L \hspace{1cm} (12)

Goal: \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} qrev(\overline{h :: t}, [k] <> [l]) = qrev(\overline{h :: t}, [k]) <> [l]. \hspace{1cm} (13)

Note that the \textit{K} and \textit{L} in (12) denote free variables and the \textit{k} and \textit{l} in the goal denote constants. Moreover, the \textit{k} and \textit{l} are annotated as sinks. Here a ripple proof requires wave-rules for \textit{qrev} and \textit{<>}, \textit{i.e.}

$$qrev(H :: T, L) \Rightarrow qrev(T, H :: L)$$ \hspace{1cm} (14)

$$H :: (T <> L) \Rightarrow (H :: T) <> L$$ \hspace{1cm} (15)

Using wave-rule (14), goal (13) ripples to give:

$$qrev(t, \overline{h :: ([k] <> [l])}) = qrev(\overline{h :: t}, [k]) <> [l]. \hspace{1cm} (16)$$

By wave-rule (15), rippling-in on the left-hand side of goal (16) gives:

$$qrev(t, [h :: k] <> [l]) = qrev(\overline{h :: t}, [k]) <> [l]. \hspace{1cm} (17)$$

To complete the rippling on the right-hand side of goal (17) a further application of wave-rule (14) is required:

$$qrev(t, [h :: k] <> [l]) = qrev(t, [h :: k]) <> [l].$$

Note that all the wave-fronts have reached sink positions, \textit{i.e.} the sink annotations identify the terms required in order to achieve a match between goal and given, \textit{i.e.}

$$qrev(t, [h :: k] <> [l]) = qrev(t, [h :: k]) <> [l].$$

Note that free variable \textit{K} within (12) is instantiated to the contents of the sink \textit{h :: k} and \textit{L} to \textit{l}. 
2.3 Rippling and Middle-out Reasoning

The final example involves a proof planning technique known as middle-out reasoning [6]. Middle-out reasoning uses meta-variables to delay choice within the search for a proof. Note that within the context of middle-out reasoning rippling is not guaranteed to terminate. Middle-out reasoning has been used successfully within program synthesis [2, 20, 19] and proof patching [13–15, 22, 18, 9, 10, 16]. Here we consider its use within the context of conjecture generalization. Proof by mathematical induction often requires a generalization step, i.e. the generalized conjecture provides a stronger induction hypothesis. Finding a suitable generalization represents an infinite branching point within the search space. The application of middle-out reasoning to the generalization problem involves using meta-variables to postulate a schematic conjecture. The instantiation of the schematic conjecture is guided by a subsequent application of the ripple proof plan. To illustrate, consider the following conjecture:

\[ \forall t \in \text{list}(\tau). \forall l \in \text{list}(\tau). \, \text{rev}(t) = \text{qrev}(t, \text{nil}) \]  

where \( \text{rev} \) denotes list reversal. A ripple guided proof of (18) will fail. The failure can be used to suggest the following schematic conjecture:

\[ \forall t \in \text{list}(\tau). \forall l \in \text{list}(\tau). \, \text{G}_1(\text{rev}(t), l) = \text{qrev}(t, F_1(l)) \]  

Note that \( F_1 \) and \( G_1 \) are second-order meta-variables. The ripple problem arising from a 1-step structural induction on \( l \) takes the form:

Given:  
\[ G_1(\text{rev}(t), L) = \text{qrev}(t, F_1(L)) \]  

Goal:  
\[ G_1(\text{rev}(\hat{h} :: t \downarrow), [l]) = \text{qrev}(\hat{h} :: \text{nil}, F_1([l])) \]  

Focusing on the left-hand side of the goal, rippling requires the following wave-rules:

\[ \text{rev}(Y :: X \uparrow) \Rightarrow \text{rev}(X) \leftrightarrow Y :: \text{nil} \]  

\[ (X \leftrightarrow Y \uparrow) \leftrightarrow Z \Rightarrow X \leftrightarrow (Y \leftrightarrow Z \uparrow) \]  

Wave-rule (22) ripples (21) to give:

\[ G_1(\text{rev}(t) \leftrightarrow \hat{h} :: \text{nil} \uparrow, [l]) = \ldots \]  

Using wave-rule (23), (24) ripples to give:

\[ \text{rev}(t) \leftrightarrow \hat{h} :: \text{nil} \leftrightarrow G_2(\text{rev}(t) \leftrightarrow \hat{h} :: \text{nil} \uparrow, [l]) = \ldots \]  

\(^3\) For more details see [14].
Note that the application of (23) instantiates $G_1$ to be $\lambda x.\lambda y.x <> G_2(x,y)$. Finally, eager matching completes the rippling on the left-hand side, instantiating $G_2$ to be a projection onto its second argument, giving:

$$\text{rev}(t) <> \left[ h :: \text{nil} <> l \right] = \ldots .$$

Finally, wave-front simplification gives:

$$\text{rev}(t) <> \left[ h :: l \right] = \ldots .$$

Turning to the right-hand side of the goal, rippling applies wave-rule (14) giving:

$$\ldots = \text{qrev}(t, h :: F_1([l])) .$$

Again eager matching completes the rippling, i.e.

$$\ldots = \text{qrev}(t, [h :: l]) .$$

Note that $F_1$ is instantiated to be the identity function, i.e. $\lambda x.x$. Now a complete match between goal and given is possible, completing the proof. The overall effect of the middle-out reasoning is to incrementally instantiate (19) to be:

$$\forall t \in \text{list}(\tau). \forall l \in \text{list}(\tau). \text{rev}(t) <> l = \text{qrev}(t, l). \quad (25)$$

As a final step we must show that (25) is indeed a generalization of (18). This means proving the conjecture

$$(\forall t \in \text{list}(\tau). \forall l \in \text{list}(\tau). \quad (\text{rev}(t) <> l = \text{qrev}(t, l))) \rightarrow (\forall t \in \text{list}(\tau). \quad (\text{rev}(t) = \text{qrev}(t, \text{nil}))).$$

Specializing $l$ to be $\text{nil}$ and simplifying the antecedent gives rise to a trivial subgoal.

### 2.4 Discussion

The presentation style adopted above exhibits two general features which we believe contribute to the difficulties some students experience when attempting to understand formal proofs. First, the fragmentation of certain aspects of the presentation. Second, the monolithic structure of the overall presentation. Below we explore these features in more detail.
**Fragmentary explanations:** Consider again the explanation presented in §2 of how a goal is annotated with respect to a given hypothesis, i.e. annotating \( (2) \) with respect to \( (1) \). Note that the goal formula is presented three times. The first shows the original goal, the second highlights the embedded skeleton, while the third identifies the wave-fronts. These three formulas represent different aspects of the one goal. We believe the fragmentation of these aspects may lead to confusion.

Another place where fragmentation arises is where meta-variables are introduced and instantiated. For example, consider schematic hypothesis \((20)\), and \((25)\), its eventual instantiation. While the relationship between \((20)\) and \((25)\) is stated in words, the reader is left to visualize the incremental instantiation of \((20)\). Again we feel the fragmentation of the information needed to understand the overall effect of the middle-out reasoning process may lead to confusion.

**Monolithic structure:** A consequence of the monolithic presentation adopted above is that all aspects of the proof are presented up-front. There is no opportunity to abstract away some of the details and observe the proof at a higher level. Likewise, the reader has no opportunity to access deeper explanations. A hierarchical style of presentation would address this problem.

Furthermore, multiple styles of presentation provide multiple perspectives onto the reasoning process. We believe that this can assist in understanding how a proof is progressing. In terms of rippling, both tree and textual presentations are typically used when introducing the notion of skeleton preservation (see Fig 1). However, a tree becomes cumbersome for a static text book style of presentation.

---

A simple 2-step outwards directed ripple is shown above. Note that both tree and textual presentations are given.

**Fig. 1.** Skeleton preservation
3 Hand-crafted Proof Animations

We now describe our proof animations and how they address the areas of difficulty highlighted in §2.4. In earlier work [8], we investigated the use of Director [21] for developing proof animations [8]. In particular, we developed sub-animations which showed the evaluation of individual method preconditions during proof construction. Our desire, however, for Web-based proof animations led us to switch to Flash [21]. We have used Flash to develop animations for the three ripple proofs presented above.

Flash is a technology developed by Macromedia initially to enable the use of vector rather than raster graphics on the Web. This idea was extended to produce a system which includes animation and interaction. Flash movies are authored in a proprietary editor and stored in a non-executable source format which can be exported to a much smaller executable form with the editor.

A Flash movie is constructed hierarchically by constructing small movies and graphical objects, and then combining them in another movie which can then in turn be used as a movie component. Sub-movies can be controlled using scripts in the time-line of the parent movie to allow them to be started and stopped at the appropriate times. A screen-shot of the Flash development environment is presented in Fig 2. Here we consider specific aspects of the animations via screen-shots. The reader is strongly encouraged to view the animations via a Flash enabled Web browser. The animations for the three ripple proofs can be accessed via:

http://www.rippling.org/

3.1 Multi-perspective and multi-layered animations

In §2.4, the merits of a multi-perspective style of presentation were discussed. Within the context of rippling, both textual and tree representations provide insights into the proof process. All our ripple proof animations include textual and tree representations, as illustrated in the snapshot given in Fig 3. A hierarchical, or layered, style of presentation was also raised as a way of addressing the monolithic nature of textual proof. We have used this idea to a certain extent in the way in which the viewer can explore rules, proof patch explanations and meta-variable instantiations. For instance, Fig 4, provides a snap-shot similar to Fig 3, in which the viewer has selected more details on the preceding step in the proof, i.e. the application of wave-rule (5). Although none of our sub-animations are particularly sophisticated, we believe this is a key strength of proof animations.

3.2 Eliminating fragmentation

The fragmentation of explanations was also identified in §2.4 as a potential source of difficulties when communicating proofs textually. The process by which a goal

---

4 See http://www.macromedia.com/software/
The development environment uses a series of panels to provide the functionality needed to edit Flash movies. The main panels needed are the drawing panel, the stage, the time-line and the library. The drawing panel (left) provides tools for creating and manipulating graphic objects similar to other drawing packages. The stage (centre) is the main page where objects can be drawn, manipulated, and arranged. The time-line (top) provides a way of organizing how the movies appear, disappear and change overtime. The library (right) provides a repository for constructed objects and movies that can be used in the movie.

Fig. 2. Flash development environment
The above screen-shot is taken from the “Outward Rippling” animation, and shows a successful outward directed ripple immediately before the given hypothesis is applied.

Fig. 3. Multi-perspective on a ripple proof
The above screen-shot is taken from the “Outward Rippling” animation, and shows how the viewer can select details on which wave-rules have been applied.

Fig. 4. Multi-layered ripple proof
formula is annotated with respect to a given hypothesis was highlighted as an example of this phenomena. We have addressed this problem through a series of overlayed animations, \textit{i.e.} we firstly used colour to highlight the embedding of the given within the goal; secondly the wave-fronts were introduced. Because, the process is presented locally, \textit{i.e.} not fragmented across a series of formulae, we believe that the process of wave-front annotations is more effectively communicated. This style of presentation is used within all three proof animations.

The instantiation of meta-variables was also highlighted as an example of the fragmentation phenomena. The middle-out ripple proof provides a good example of this phenomena. We tackled the problem by simply ensuring that all occurrences of a meta-variable are simultaneously instantiated during the course of an animation. For example, in Fig 5 two snap-shots from the middle-out ripple proof animation are shown. The first is before $G_1$ is instantiated while the second shows the proof state after the partial instantiation of $G_1$. Although not shown in Fig 5, the viewer can select to see the definition of meta-variables during the animation via the buttons on the right-hand side of the canvas.

Proof animations are not without their limitations. A problem with animating the transitions between snap-shots of a proof is that they often correspond to proof steps that can be considered to be atomic. So that there is no sensible way to decompose the steps that would make sense in an animation. Consequently, the intermediate stages which are necessary to depict movement are not themselves valid stages in the proof, but merely there as a presentational mechanism. It could be argued that showing movement reveals the ideas behind the proof but it could also be argued that these intermediate stages detract from presentation of the proof as rigorous. There is not really any way around this problem except to simply make sure that the animation does not place emphasis on any intermediate stages and accept them as just a way of presenting the transition of the goal from one state to the next.

4 \hspace{1em} A Proposal for Automated Proof Animations

As demonstrated by our rippling animations, Flash is a powerful tool for developing proof animations. Ideally we would like to be able to automate the generation of such proof animations directly from a proof planner. However, the hand-crafting required in developing a Flash animation precludes this possibility. We believe that the programmability of Scalable Vector Graphics (SVG) [1] provides the best opportunity for us to achieve our goal of automated proof animations. Below we propose an extension to the proof planning technique that would support the automatic generation of proof animations using SVG.

Proof planning builds upon the LCF style of theorem proving [11], where primitive proof steps are packaged-up into programs known as \textit{tactics}. Starting with a set of general purpose tactics, plan formation techniques are used to construct a customized tactic for a given conjecture. The search for a customized tactic is constrained by a set of \textit{methods}, each of which specifies the applicability of a general purpose tactic in terms of preconditions. Collectively a set of methods
The above screen-shots are taken from the “Generalization” animation, and show a middle-out ripple proof before and after the instantiation of meta-variable $G_1$. Note that the viewer can also select details of meta-variable instantiations via the buttons on the right-hand side of the screen.

Fig. 5. Meta-variable instantiation
defines a proof plan. Where the application of a proof plan fails, proof critics [12, 14] are used to automatically analyse and generate proof patches.

We propose to extend both the definition of methods and critics to include an animation slot. The animation slot will contain parameterized SVG. The basic idea is that method application will yield a customized animation as well as a customized tactic. Similarly, critic application will yield a customized animation as well as a customized patch. Moreover, the structure of the animation slot will reflect the structure of the method (critic). That is, the animation slot will contain a parameterized animation corresponding to the overall effect of the method (critic). Parameterized animations for each method (critic) precondition will also be included. As a consequence, a method (critic) application can be viewed at different levels of detail, providing multi-layered animations. Method hierarchies will obviously bring additional layering to proof animations.

As well as attaching animations to methods and critics, we also propose that the various objects available to the proof planner will also be associated with animations, e.g. conjectures, induction schemas, rewrite rules, etc. Moreover each classification of rewrite rule known to the proof planner will also have its own form of animation. Note that rewrite rules fit well with the generation of SVG animations, i.e. an animation is defined in terms of initial and final states. Again these animations will be parameterized, and instantiation will take place through method application, or goal initialization in the case of a conjecture. We envisage that the generation of these “small animations” as a task that can be mechanized.

During proof planning, certain information is lost from the resulting tactic, e.g. the fact that a lemma is generated during proof planning is not obvious by inspecting the tactic alone. We envisage, therefore, the creation of a story-board during the search for a proof. Having a story-board will be similar to having an audit trail of the proof search. A post proof planning process will then flesh-out the story-board using the animation fragments associated with the method and critic applications, as well as the animations associated with rules etc. Turning the story-board into a complete proof animation represents a non-trivial planning problem.

5 Formative Feedback via Animation

We are currently investigating the potential role that proof animation can play in providing formative feedback to students. Think of a scenario where a student provides text based proofs to formal reasoning problems. Given an appropriate proof plan, a proof planner could then be used to assess the correctness and quality of student proofs. Moreover, where a bug is found within a proof we envisage the use of animation in communicating the nature of the bug and potential suggestions as to how a student might patch their reasoning. Interactive proof critics [17] provide a natural starting point. The degree of assistance given to a student could be varied. For instance, an animation could be generated simply based upon the critics analysis of the bug. Alternatively, for a weaker student a series of steps could be presented, i.e. intermediate milestones on the
road to a complete proof. These steps would be generated behind the scenes via proof patching. The number of steps presented could be varied, and ultimately the gaps between steps could be bridged through proof animation.

6 Conclusion

We have argued for a dynamic style of proof presentation where proof planning is animated on the Web. While our current animations are hand-crafted, we have outlined a proposal for automating the construction of proof animations. We see a potential role for proof animation within the context of formative feedback. Our evidence for the merits of the work is purely anecdotal, i.e. it is based upon informal feedback from students. A formal user study is therefore part of our future work plan.

Acknowledgements

The research reported in this paper is supported by EPSRC grant GR/S01771. We would like to thank Alan Bundy, Bill Ellis, Benjamin Gorry and members of the Dependable Systems and Mathematical Reasoning Groups for their feedback on this work.

References


