THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

'The Tinker' for Rodin

Citation for published version:
Liang, Y, Lin, Y & Grov, G 2016, 'The Tinker' for Rodin. in *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings.* Lecture Notes in Computer Science, vol. 9675, Springer, Cham, pp. 262-268, Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, Linz, Austria, 23/05/16. https://doi.org/10.1007/978-3-319-33600-8_19

Digital Object Identifier (DOI):
10.1007/978-3-319-33600-8_19

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 28. Feb. 2021

'The Tinker' for Rodin*

Yibo Liang[1], Yuhui Lin[1] and Gudmund Grov[1]

Heriot-Watt University, Edinburgh, UK {yl9, Y.Lin, G.Grov}@hw.ac.uk

Abstract. *PSGraph* [3] is a graphical proof strategy language, which uses the formalisation of labelled hierarchical graphs to provide support for the development and maintenance of large and complex proof tactics. PSGraph has been implemented as the *Tinker* system, which previously supported the *Isabelle* and *ProofPower* theorem provers [4]. In this paper we present a *Rodin* version of Tinker, which allows Rodin users to encode, analyse and debug their proof strategies in Tinker.

1 PSGraph & Tinker

PSGraph [3] is a graphical proof strategy language, where proof tactics are represented as directed hierarchical graphs, which provides an intuitive representation to understand and work with proof strategies. The nodes in PSGraph contain tactics, provided by the underlying theorem prover, or nested graphs, and are composed by labelled wires. The labels are called *goal types* which are predicates describing expected properties of sub-goals. Each sub-goal is a special *goal node* on the graph, which "lives" on a wire. Evaluation is handled by applying a tactic to a goal node that is on one of its input wires. The resulting sub-goals are sent to the out wires of the tactic node. To add a goal node to a wire, the goal-type must be satisfied. This mechanism is used to ensure that goals are sent to the right place.

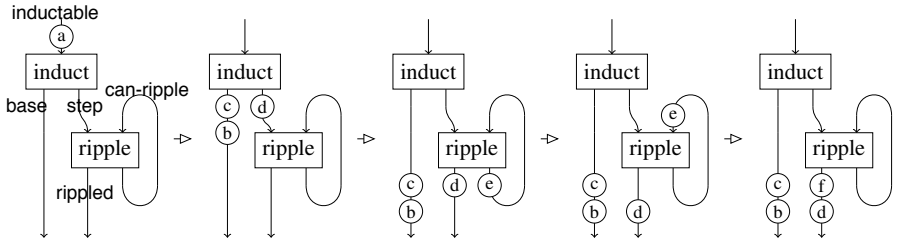

Fig. 1. An example of using PSGraph [3]

In Fig. 1 we show an example adapted from [3]. This is a proof strategy called *rippling*, which has also been used to automate invariant proofs in Event-B [5].

* This work has been supported by EPSRC grants EP/J001058, EP/K503915, EP/M018407 and EP/N014758.

Here, there are two tactics nodes, *induct* and *ripple*, connected by edges with suitable goal types to guide the sub-goals to the correct target. In this example, a goal node, labelled by a is the input of the first tactic node (*induct*). Applying the *induct* tactic to a generates three new sub-goals, where a and b are *base* cases, while d is a *step* case. The step cases will be sent to the *ripple* tactics. Over two iterations, the *ripple* tactic will generate two new sub-goals, f and d, which will be sent to the output of the graph. Major features of PSGraph is: this ability to control the goal flow with goal types; step through evaluation during debugging as Fig. 1 illustrates and abstraction of the order and number of sub-goals (e.g. both b and c were sent to the same edge).

PSGraph is formalised using *string diagrams* [1], which supports dangling edges (i.e. an edge without a source/target): an edge without a source becomes the input; and an edge without a target becomes an output. By exploiting the goal types, string diagrams support a novel type-correct composition of tactics[1]. For example, we can only connect the (output) *base* edge of Fig. 1 with a graph that has an input *base* goal type.

PSGraph has been implemented in the *Tinker* tool [4,7], which also provide support for developing, debugging and maintaining proof strategies. We will illustrate this in the tool in §2.

Previously Tinker only supported *Isabelle* and *ProofPower*. In this paper, we present a *Rodin* version of Tinker, which consists of the Tinker tool and a 'Tinker for Rodin' plugin. This will enable Rodin users to encode their proof strategies in Tinker with the following features: (1) users can draw proof strategies as flow graphs; (2) users can reuse and modify existing proof strategies by drawing; (3) users can step through how sub-goals flow through the graph during a proof, including debugging features such as breakpoints and the ability to modify the proof strategy. In §2 we will show how Tinker can be used to support developing and debugging proof strategies with a case study. This is followed by a discussion of the architecture and implementation details in §3. We conclude and briefly discuss further work in §4.

2 Developing and debugging proof strategies in Rodin

To illustrate usage of the tool, consider proof obligations (POs) `X_act/inv19` (1) and `X_act/inv21` (2) taken from an encoding of a landing gear case study [8].

$$\begin{array}{l} \texttt{partition}(\texttt{EV}, \{prs_e v\}, \{dpr_e v\}, \{opn_e v\}, \{cls_e v\}, \{ext_e v\}, \{rtr_e v\}, \{no_e v\}), \ldots \\ \quad \vdash N(ext \mapsto which\_ev) \in \{prs\_ev, opn\_ecv\} \ \wedge \ ext = \texttt{FALSE} \\ \qquad \Rightarrow \ X(rtr\_ev = \texttt{TRUE}) \end{array} \tag{1}$$

$$\begin{array}{l} \texttt{partition}(\texttt{EV}, \{prs_e v\}, \{dpr_e v\}, \{opn_e v\}, \{cls_e v\}, \{ext_e v\}, \{rtr_e v\}, \{no_e v\}), \ldots \\ \vdash N(ext \mapsto which\_ev) \in \{rtr\_ev, ext\_ecv\} \ \Rightarrow \ X(opn\_ev = \texttt{TRUE}) \end{array} \tag{2}$$

Both of these require interactive proofs. The proof strategy for `X_act/inv19` is as follows:

[1] Composition of two graphs is formalised as a categorical push-out [1].

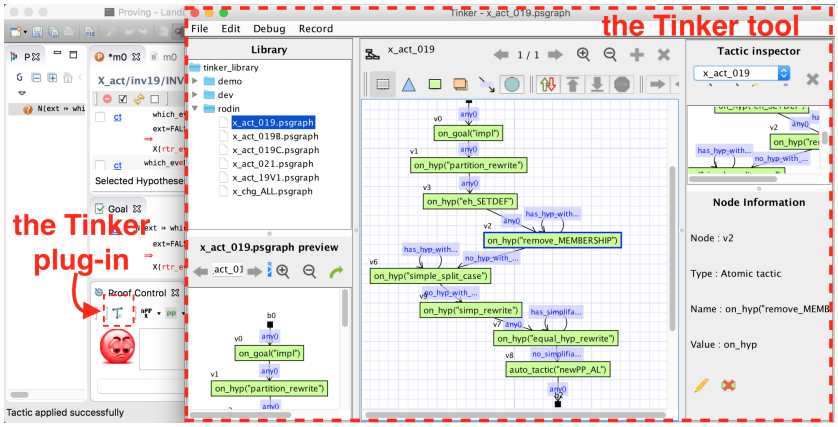

Fig. 2. The Tinker plug-in and Tinker GUI

- Eliminate the implication ($\Rightarrow$) in the goal;
- Unfold the definitions of partition (`partition`);
- Rewrite hypotheses with the set equality hypothesis unfolded from `partition`;
- Unfold the definitions of membership ($\in$);
- Apply a case split on any disjunctive hypothesis;
- Simplify hypotheses with the default rewrite rules of Rodin;
- Simplify hypotheses by rewriting with equational hypotheses;
- Apply the auto prover on the remaining sub-goals.

To develop this proof strategy, users can open the Tinker tool (as seen in Fig. 2) and draw it in a click and drag style. The PSGraph we have developed is shown in Fig. 3 (left), where each green box is a proof tactic provided by Rodin:

on_goal means that the given rewrite rule(s) should be applied to the goal, e.g. `on_goal(impI)` eliminates the implication ($\Rightarrow$) in the goal.

on_hyp means that the given rewrite rule(s) should be applied to the hypotheses, e.g. `on_hyp(remove_membership)` rewrites any hypotheses of the shape

$$x \in \{S_1, \ldots, S_n\}$$

to

$$x = S_1 \vee \cdots \vee x = S_n$$

auto_tactic to apply the automatic tactic specified by the argument, e.g. `auto_tactic(newPP_AL)` calls the interface of Rodin's `newPP_ALL`.

A key feature of Tinker is that the edges of the graph are labeled by *goal types*, which are predicates that are used to guide sub-goals to the correct target. This is also used to pinpoint where a proof fails during debugging. Here, we illustrate some of the goal types that we have developed for Rodin proofs:

any is the default goal-type which allows any goal to proceed.

(has/no)_hyp_with_topsymbol succeeds if there is (or is not) a hypothesis in which the top level symbol is the operation specified by the argument.

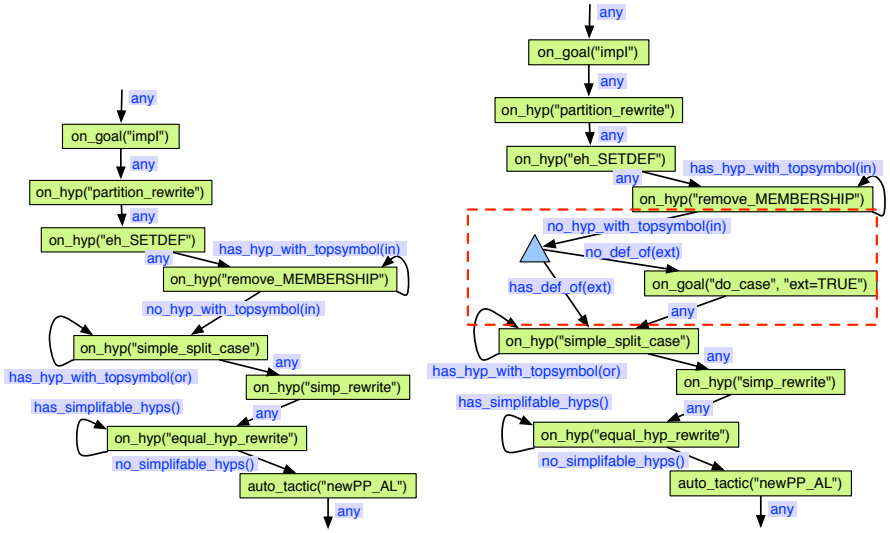

Fig. 3. Proof strategy of `X_act/inv19` (left) and `X_act/inv21` (right)

(has/no)_simplifable_hyps succeeds if there exists (or not exists) any equational hypothesis to rewrite the hypotheses.

The available tactics and goal-types are hard-coded in the current implementation, meaning users cannot define new ones. We discuss this limitation further in §4.

Tinker allows users to save the existing proof strategies into a library and import those from the library to develop new ones. To illustrate, the proof of `X_act/inv21` only differs from `X_act/inv19`, in that a case split on `ext=TRUE` is required before applying case split on the disjunctive hypothesis. Fig. 3 (right) shows the proof strategy which is developed based on the `X_act/inv19` one, with the changes highlighted. Tinker could then have been used to help generalising these strategies into a single more high-level strategy.

To apply a proof strategy to a PO, users can open the PO in Rodin and then click the button from the *prove control view* in Rodin. After selecting the proof strategy to be applied, Tinker will be launched to allow users to step through proofs with the features, such as interactive controlled inspection, debugging using breakpoints and a logging mechanism [7]. A screenshot is shown in Fig. 2. The demo of applying the proof strategies to the two POs are available in [9].

3 Implementation

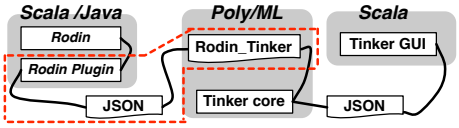

Fig. 4. Architecture

The Rodin version of Tinker consists of three parts: the **GUI**, **CORE** and the **Rodin plug-in**. Each is shaded in a separated grey box, and the new parts, which are contributions of this paper, are highlighted with dotted lines in Fig. 4. The GUI is implemented in Scala and provides a graphical way of developing, and debugging proof strategies. The core is implemented in Poly/ML, and holds the key functionalities such as goal-type checking and evaluation. The core is theorem prover independent, as most functionality is implemented using ML `functor`s. Each theorem prover, i.e. Isabelle, ProofPower and Rodin, has a prover configuration `structure` that implements a provided `signature`, as indicated by `Rodin_Tinker` for Rodin in Fig. 4.

Isabelle and ProofPower are both encoded in Poly/ML which made it easy to integrate with the core. For Rodin, we had to develop a JSON-based communication protocol between the actual Rodin plug-in and the Tinker core. The Rodin plug-in therefore acts as the communication agent between the *Rodin Proof Obligation Manager (POM)* and the core. It is responsible for calling the correct tactic in the POM and send the updated proof status back to the core. To illustrate, the tactic application

```
on_goal(impI)
```

is translated to a JSON message containing the command

```
APPLY_TACTIC
```

and the parameters `ON_GOAL` and `impI`. This is sent to the Rodin plug-in, which will interpret the message to call the following interface in Rodin to eliminate the implication in the goal:

```
Tactics.impI().apply(pnode, pm)
```

where `pnode` is the proof tree node at which this tactic should be applied and `pm` is the proof monitor in Rodin.

4 Conclusion & future work

We have presented a Rodin version of the Tinker tool. This is the first Tinker supported theorem prover that does not strictly follow the LCF approach [2] and that required an additional communication protocol. The plug-in allows Rodin user to develop and debug tactics as graphs, and the approach has been shown to scale to an industrial setting [6].

We believe that the integration of Rodin and Tinker has great potential. To illustrate, a tactic node in a PSGraph can be used to select relevant hypothesis, which are then used as parameters for subsequent tactic node.s For example, we can apply a tactic

```
bind_hyp_with_topsymbol (in, ?x)
```

which selects all the hypotheses satisfying that the top symbol is $\in$, and then bind the list of the hypotheses to an environment variable `?x`. We can then apply a tactic with the variable, e.g.

```
on_hyp (remove_MEMBERSHIP, ?x)
```

which will apply the tactic to eliminate $\in$ for the list hypotheses bound in `?x`. This feature, which seems very useful for Rodin which can have a large number of hypothesis, is supported in the Isabelle and ProofPower version, and we need to update the Rodin prover configuration `structure` to support it. We also would like to support configurable tactic and goal type features that will allow users to define new tactics and goal types in a easy manner without resorting to the source code. We would also like to investigate richer proof strategies in Rodin. For example, the second author's PhD thesis used rippling (in Isabelle) to automate Rodin POs [5]. We have previously developed a simplified version of rippling in Tinker [4] which we would like to incorporate with the Rodin version.

References

1. Lucas Dixon and Aleks Kissinger. Open graphs and monoidal theories. *CoRR*, abs/1011.4114, 2010.
2. M.J. Gordon. Edinburgh LCF: a mechanised logic of computation. 1979.
3. G. Grov, A. Kissinger, and Y. Lin. A graphical language for proof strategies. In *LPAR*, pages 324–339. Springer, 2013.
4. G. Grov, A. Kissinger, and Y. Lin. Tinker, tailor, solver, proof. In *UITP 2014*, volume 167 of *ENTCS*, pages 23–34. Open Publishing Association, 2014.
5. Y. Lin. *The Use of Rippling to Automate Event-B Invariant Preservation Proofs*. PhD thesis, 2015.
6. Y. Lin, O'Halloran C. Grov, G. and, and P. G. A super industrial application of PSGraph. In *ABZ 2016*, to appear.
7. Y. Lin, P. Le Bras, and G. Grov. Developing & debugging proof strategies by tinkering. In *TACAS 2016*, to appear.
8. W. Su and JR Abrial. Aircraft landing gear system: approaches with Event-B to the modeling of an industrial system. In *ABZ 2014*.
9. Y.Liang Y. Lin and G. Grov. Tinker - ABZ 16 paper ressources. `http://ggrov.github.io/tinker/abz2016/`. Accessed: 2016-2-3.