Diabelli: A Heterogeneous Proof System*

Matej Urbas and Mateja Jamnik

Computer Laboratory, University of Cambridge, UK
{Matej.Urbas,Mateja.Jamnik}@cl.cam.ac.uk

Abstract. We present Diabelli, a formal reasoning system that enables users to construct so-called heterogeneous proofs that intermix sentential formulae with diagrams.

1 Introduction

Despite the fact the people often prove theorems with a mixture of formal sentential methods as well as the use of more informal representations such as diagrams, mechanised reasoning tools still predominantly use sentential logic to construct proofs – diagrams have not yet made their way into traditional formal proof systems. In this paper, we do just that: we present a novel heterogeneous theorem prover Diabelli that allows users to seamlessly mix traditional logic with diagrams to construct completely formal heterogeneous proofs – Fig. 1 shows an example of such a proof. In particular, Diabelli connects a state-of-the-art sentential theorem prover Isabelle [1] with our new formal diagrammatic theorem prover for spider diagrams called Speedith [2]. The interactive user interface allows for displaying typical sentential proof steps as well as visual diagrammatic statements and inferences. The derived heterogeneous proof is certified to be (logically) correct. Our heterogeneous framework is designed to allow reasonably easy plugin of other external proof tools (sentential or diagrammatic), thus potentially widening the domain of problems that can be tackled heterogeneously.

The motivation for our work is not to produce shorter and faster proofs, but to provide a different perspective on formulae as well as to enable a flexible way of proving them. The users can switch representation and type of proof steps at any point, which gives them options and freedom in the way they construct the proof, tailored to their level of expertise and their cognitive preference. The integration in Diabelli's framework benefits both: diagrammatic reasoners gain proof search automation and expressiveness of sentential reasoners, while sentential reasoners gain access to another, diagrammatic view of the formula and its proof which might provide a better insight into the problem.

2 Heterogeneous Reasoning Components

The kind of problems that can be tackled in our Diabelli heterogeneous framework depends on the choice of sentential and diagrammatic reasoners, thus on the domains of Isabelle and Speedith. Isabelle, as a general purpose theorem prover, covers numerous different domains. Speedith's domain is monadic first-order logic with equality

* Supported by EPSRC Advanced Research Fellowship GR/R76783 (Jamnik), EPSRC Doctoral Training Grant and Computer Lab Premium Research Studentship (Urbas).

B. Gramlich, D. Miller, and U. Sattler (Eds.): IJCAR 2012, LNAI 7364, pp. 559–566, 2012.

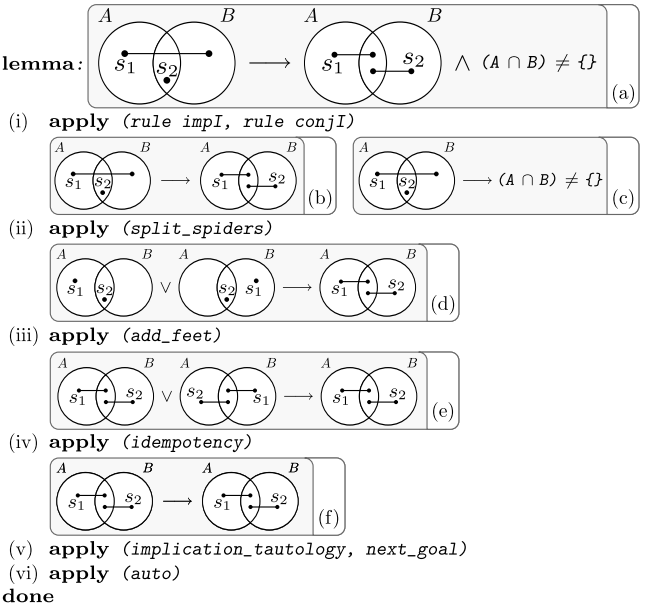

Fig. 1. A heterogeneous proof of a statement with sentential steps (i), (v), (vi) and diagrammatic steps (ii), (iii), (iv). (i) splits the lemma into a diagrammatic and a sentential sub-goal. (ii), (iii) and (iv) prove the diagrammatic sub-goal. (v) discharges the goal (f) and proceeds to the sentential goal in (c), which is discharged by *auto* in (vi) – a powerful proof tactic in Isabelle. All steps are verified by Isabelle's logical kernel.

(MFOLE). Thus, heterogeneously, in Diabelli system we can currently prove all theorems that contain subformulae of MFOLE provided they are expressed with spider diagrams. Even though this domain is decidable, so potentially Isabelle alone could sententially prove any theorem of MFOLE, our motivation lies elsewhere, namely in heterogeneous proofs and in showing the feasibility of formal heterogeneous reasoning, rather than in sentential proofs alone. Spider diagrams are a case study and a prototype diagrammatic language for the Diabelli framework. When Diabelli is extended with new theorem provers, the domain of problems covered will extend accordingly.

2.1 The Diagrammatic Reasoner

Spider Diagrams. The language of spider diagrams is used in Speedith, which constitutes the diagrammatic reasoning part of our heterogeneous system. It is equivalent to MFOLE, and is sound and complete (see [3] for details).

Spider diagrams are composed of *contours*, *zones*, *spiders*, *shading*, and *logical connectives*. *Contours* are closed curves which represent sets and assert relationships between them through relative spatial positioning. For example, the enclosure of one contour in another denotes the subset and superset relations. Fig. 1(c) shows a diagram containing two contours named with *labels* A and B. The set of all contour labels in a diagram d is denoted by $L(d)$.

A *zone* is an area lying within some (or no) collection of contours and lying outside the others. It denotes a set and is represented as a pair of finite, disjoint sets of contour labels (in, out), such that $in \cup out = L(d)$. A zone (in, out) lies within contours in and outside of contours out. The diagrams in Fig. 1(c) contains the zones $(\emptyset, \{A, B\})$, $(\{A\}, \{B\})$, $(\{B\}, \{A\})$, and $(\{A, B\}, \emptyset)$. The set of zones in a diagram is denoted by $Z(d)$. The zones in $Z(d)$ can be *shaded*, denoted by $ShZ(d)$. Shading indicates that the zone's only elements are the spiders, which places an upper bound on the cardinality of the set.

Spiders are connected acyclic graphs whose nodes (*feet*) lie in a set of zones. A spider asserts the existence of an element in the union of the zones where its feet reside. The set of all spiders in a diagram d is denoted by $S(d)$, and the function $\eta : S(d) \rightarrow \mathcal{P}(Z(d)) \setminus \emptyset$ returns the *habitat*, that is, the set of zones in which a spider's feet are placed. The diagrams in Fig. 1(c) contains two spiders s_1 and s_2, with s_1, for example, having the habitat $\{(\{A\}, \{B\}), (\{B\}, \{A\})\}$.

The diagrams considered so far are called *unitary* diagrams. Spider diagrams can be negated with the $\neg$ operator, and joined with binary logical connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), and $\Leftrightarrow$ (equivalence) into *compound diagrams* (e.g., Fig. 1(b)). For a complete formal specification of the semantics of spider diagrams, see [2]. In Isabelle/HOL we formalise it with the `sd_sem` interpretation function (Sec. 3.1).

Diagrammatic Inference Rules. Spider diagrams are equipped with inference rules [2,3] that are all proved to be sound, hence proofs derived by using them are guaranteed to be correct. Step (iii) in Fig. 1 from diagrams in (d) to (e) shows an application of the diagrammatic inference rule *add feet* (which adds feet to an existing spider to assert that it could live in another region too). Speedith allows interactive application of this and a number of additional inference rules (see [2] for complete specification).

Speedith's Architecture. Speedith [2] is our implementation of an interactive diagrammatic theorem prover for spider diagrams. It has four main components:

1. abstract representation of spider-diagrammatic statements,
2. the reasoning kernel with proof infrastructure,
3. verification of diagrammatic proofs, including input and output system for importing and exporting formulae in many different formats, and
4. visualisation of spider-diagrammatic statements.

The abstract representation of spider diagrams is used internally in Speedith to represent all spider-diagrammatic formulae (see [2] for details). Speedith can be used as a standalone interactive proof assistant, but it can also be easily plugged into other systems via its extensible mechanism for import and export of spider diagrams. Currently, spider diagrams can be exported to Isabelle/HOL formulae or a textual format native to Speedith; and for import, MFOLE formulae need to be translated to Speedith's native textual format.

2.2 The Sentential Reasoner - Isabelle

For the sentential part of our heterogeneous reasoning framework, we chose Isabelle, which is a general purpose interactive proof assistant. This choice was arbitrary, any

other highly expressive and interactive theorem prover could be used. In particular, Diabelli requires the reasoner to provide a way to interactively enter proof goals and proof steps. The reasoner should also be able to output its current proof goals and incorporate new proof steps from external tools. In the future we could add a requirement that the reasoner should support storage of arbitrary data – this could be utilized when a diagrammatic language is not translatable into a reasoner's format.

3 Integration of Diagrammatic and Sentential Reasoners

The Diabelli framework integrates diagrammatic and sentential provers on two levels. Firstly, it connects them via drivers that in case of Speedith and Isabelle contain a bidirectional translation procedures and a formal definition of the semantics of diagrams – this is presented next. Secondly, the interactive construction of heterogeneous proofs is facilitated through Diabelli's graphical user interface, which is presented in Sec. 4.

3.1 Interpretation of Spider Diagrams in Isabelle/HOL

To formally define the semantics of spider diagrams we specify a theory of spider diagrams in Isabelle/HOL. In particular, we provide a formalisation of the abstract representation and its interpretation, which verifies that our encoding is faithful to the formalisation in [3]. The main part of this theory is the function `sd_sem` (Def. 2) which translates the abstract representation of spider diagrams to Isabelle/HOL formulae. The function `sd_sem` interprets a data structure `SD` (Def. 1), which closely matches the abstract representation of spider diagrams.

Definition 1. *The* `SD` *data structure captures the abstract representation of spider diagrams and is defined in Isabelle/HOL as:*

```
datatype SD = NullSD
            | CompoundSD {operator: bool =>...=> bool, args: sd list}
            | UnitarySD {habitats: region list, sm_zones: zone set}
```

The unitary diagram contains a list of regions (spider habitats that η generates) and a set of zones. Regions are sets of zones, zones are pairs of in- and out-contours, and contours are native Isabelle/HOL sets. The data structure `SD` does not contain a list of spider names S – they are generated by the interpretation function `usd_sem` (see Def. 3).

Definition 2. *The* `sd_sem` *function takes as an argument a description of the spider diagram and produces a FOL formula that corresponds to the meaning of the given spider diagram. The function is defined in Isabelle/HOL as:*

```
fun sd_sem (spider_diagram : SD) =
    NullSD -> True
  | CompoundSD operator args -> apply operator (map sd_sem args)
  | UnitarySD habitats sm_zones -> usd_sem habitats sm_zones
```

The function `sd_sem` formally specifies the semantics of spider diagrams. The *null spider diagram* is interpreted as the logical truth constant. The *compound spider diagrams* are interpreted as a composition of a number of spider diagrams with `operator`. The interpretation of a *unitary spider diagram* is central to the specification and is defined by the function `usd_sem` in Def. 3.

Definition 3. *The* `usd_sem` *function interprets a unitary spider diagram which is given through a list of regions (i.e., habitats of all spiders) and a set of shaded zones. It produces a FOL formula that describes the meaning of the unitary spider diagram:*

```
fun usd_sem (habs: region list, sm_zones: zone set) =
    for each h in habs
             conjunct ∃s. s ∈ ⋃_{zone∈h} sd_zone_sem zone
             spiders ← spiders ∪ s
    ∧ distinct spiders
    ∧ ∀z ∈ sm_zones. sd_zone_sem z ⊆ spiders
```

where we use this shorthand: (**for each** *x* **in** *[x1...xn]* **conjunct** *P(x)*) ≡ *P(x1)∧...∧ P(xn)*. We define *sd_zone_sem* as:

fun *sd_zone_sem* (in, out) = [⋂$_{c\in in}$ *set_of* c] \ [⋃$_{c\in out}$ *set_of* c]

where in and out are the sets of in- and out-contours of the given zone. For every habitat h, *usd_sem* introduces a fresh existentially quantified variable s (a spider) and asserts that s lives in the region defined by h. Once all variables s_i are introduced, they are declared distinct and shaded zones are interpreted as sets that may only contain spiders.

Here is an example of how *sd_sem* interprets the spider diagram from Fig. 1(b):

```
(i)   sd_sem CompoundSD{operator →,
                        [UnitarySD{[[(A,B),(B,A)],[(AB,∅)],{}},
                         UnitarySD{[[(A,B),(AB,∅)],[(B,A),(AB,∅)]],{}}]}
(ii)  (usd_sem UnitarySD{[[(A,B),(B,A)],[(AB,∅)],{}}) →
      (usd_sem UnitarySD{[[(A,B),(AB,∅)],[(B,A),(AB,∅)]],{}})
(iii) (∃s1.s1∈(sd_zone_sem(A,B) ∪ sd_zone_sem(B,A)) ∧
       ∃s2.s2∈ sd_zone_sem(AB,∅) ∧ distinct[s1,s2]) →
      (∃s1.s1∈(sd_zone_sem(A,B) ∪ sd_zone_sem(AB,∅)) ∧
       ∃s2.s2∈(sd_zone_sem(B,A) ∪ sd_zone_sem(AB,∅)) ∧ distinct[s1,s2])
(iv)  (∃s1.s1∈(A\B ∪ B\A) ∧ ∃s2.s2∈(A∩B) ∧ distinct[s1,s2]) →
      (∃s1.s1∈(A\B ∪ A∩B) ∧ ∃s2.s2∈(B\A ∪ A∩B) ∧ distinct[s1,s2])
```

where step (i) is a call to the *sd_sem* function, which applies the operator and calls the function *usd_sem* in step (ii). *usd_sem* existentially quantifies variables with their regions in step (iii). Lastly, in step (iv), *sd_zone_sem* interprets zones as sets, which produces the final formula.

3.2 Translation of Isabelle/HOL to Spider Diagrams

Above, we showed how diagrams are translated to MFOLE expressions via *sd_sem* function. We now give a translation in the other direction: from MFOLE expressions in Isabelle/HOL to spider diagrams.

An algorithm for conversion from MFOLE formulae to spider diagrams exists, but it was shown to be intractable for practical applications [4]. Consequently, Diabelli currently translates formulae that are in a specific form, called SNF (*spider normal form*), which is based on the *sd_sem* function. Whilst SNF is a syntactic subset of MFOLE, it is important to note that it is able to express any spider-diagrammatic formula. An example of an SNF formula of a compound diagram was given in line (iv) of the translation example in Sec. 3.1 above.

The translation procedure recursively descends into Isabelle's formulae, which are internally represented as trees, and returns the abstract representation of the corresponding spider diagram by essentially reversing the `sd_sem` function. A future goal is to extend this translation with heuristics that would cover a wider range of formulae.

4 Architecture of Diabelli

The architecture of Diabelli heterogeneous framework with Isabelle and Speedith plugins is illustrated in Fig. 2. Diabelli utilizes a plugin system of the I3P framework [5]

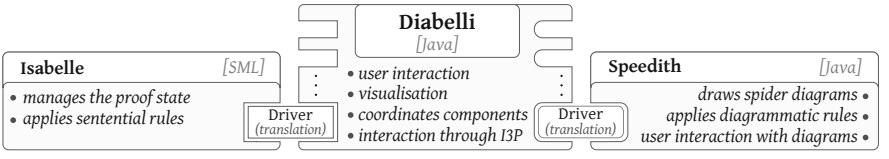

Fig. 2. The architecture of the Diabelli framework with Isabelle and Speedith

to connect the user interfaces of the sentential and diagrammatic provers. Fig. 3 shows Diabelli's graphical user interface. User commands are passed from I3P to Isabelle to

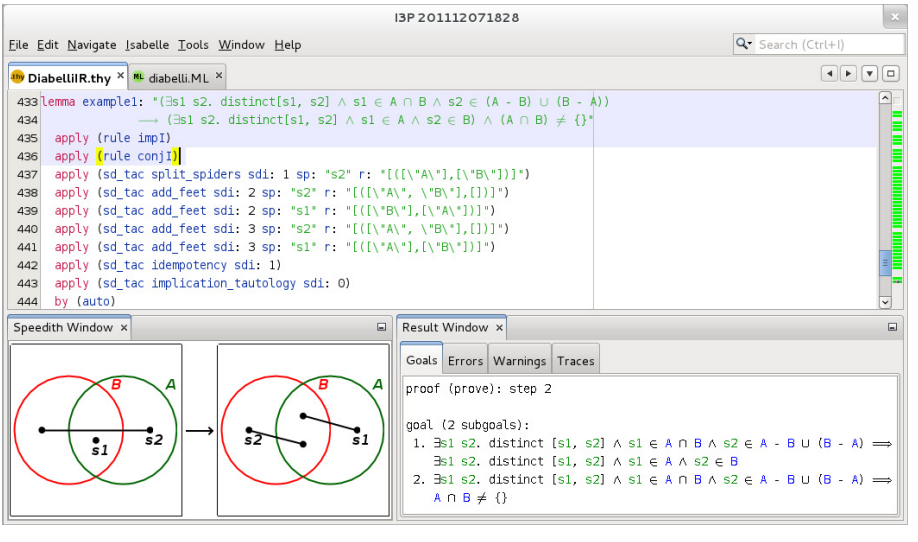

Fig. 3. A screenshot of a heterogeneous proof in Diabelli

execute them. The results are returned back to I3P, which displays them in a separate result window. The commands are read from a user-edited theory file, which may contain, for example, custom definitions, lemmas, and proof scripts. Diabelli's users may

instruct I3P step by step to issue the sentential commands to Isabelle or the diagrammatic instructions to Speedith.

Diagrammatic instructions take the same textual form in the theory file as any other Isabelle tactic (see the *sd_tac* entries in Fig. 3). These instructions, however, are not manually written by the user, but are generated by Diabelli through the user's point-and-click interactions with the diagram (note the Speedith sub-window in Fig. 3).

Diabelli automatically presents every translatable sentential sub-goal as a spider diagram in the Speedith window. If the user chooses to apply a diagrammatic inference rule on (part of) it, then Speedith executes the rule and produces a transformed diagram, which it passes back to Diabelli to replace the old sub-goal.

Diabelli currently connects only one general purpose theorem prover with a single diagrammatic language and its logic. However, we designed the Diabelli framework to be easily extended with new diagrammatic or sentential languages and logical systems by leveraging on the extensibility of the I3P framework that can manage multiple provers at the same time. Formalising the requirements for Diabelli plugins remains part of our future work.

5 General Observations

Diabelli is a novel heterogeneous reasoning framework that is a proof of concept for the connection of more powerful provers. It demonstrates how sentential and diagrammatic theorem provers can be integrated into a single heterogeneous framework. We show that using these to interactively reason with mixed diagrammatic and sentential inference steps is feasible and formally verifiable – this is breaking new ground in mechanised reasoning. Diabelli provides an intuitive interface for people wanting to understand the nature of proof. It more closely models how humans solve problems than existing state-of-the-art proof tools, is adaptable and flexible to the needs of the user, and capitalises on the advantages of each individual proof system integrated into Diabelli.

Applying either or both, diagrammatic and sentential proof steps is seamless. The normal workflow of Isabelle is not modified by the diagrammatic subsystem, moreover, the diagrammatic steps may be applied whenever a sentential formula can be translated into the diagrammatic language.

Closest to Diabelli is the Openproof [6] framework which is the only other existing system that facilitates the construction of heterogeneous reasoning systems. However, it does not integrate existing reasoning systems, but rather provides a way of combining different representations and logics.

Diabelli is implemented in SML and Java; its sources are available from `https://gitorious.org/speedith`. With Diabelli, we can heterogeneously prove all theorems that contain subformulae of MFOLE expressed with spider diagrams – this is a significant range and depth of theorems.

Despite our focus on the language of spider diagrams, Diabelli introduces a way to extend its scope to other domains. It is designed as a plugable system for seamless integration of other diagrammatic and sentential theorem provers – we are currently developing drivers for other systems to demonstrate the scalability of the Diabelli framework. A future direction is to establish a formal and concise specification of the plugin interface required to add new systems that extend Diabelli's problem domain.

References

1. Paulson, L.C.: Isabelle - A Generic Theorem Prover. LNCS, vol. 828. Springer, Heidelberg (1994)
2. Urbas, M., Jamnik, M., Stapleton, G., Flower, J.: Speedith - a diagrammatic reasoner for spider diagrams. In: Diagrams. LNCS. Springer (2012)
3. Howse, J., Stapleton, G., Taylor, J.: Spider Diagrams. LMS JCM 8, 145–194 (2005)
4. Stapleton, G., Howse, J., Taylor, J., Thompson, S.J.: The expressiveness of spider diagrams. JLC 14(6), 857–880 (2004)
5. Gast, H.: Towards a Modular Extensible Isabelle Interface. In: 22nd TPHOLs, pp. 10–19. TU Muenchen, Institut fuer Informatik (2009)
6. Barker-Plummer, D., Etchemendy, J., Liu, A., Murray, M., Swoboda, N.: Openproof - A Flexible Framework for Heterogeneous Reasoning. In: Stapleton, G., Howse, J., Lee, J. (eds.) Diagrams 2008. LNCS (LNAI), vol. 5223, pp. 347–349. Springer, Heidelberg (2008)