A Framework for Heterogeneous Reasoning in Formal and Informal Domains

Matej Urbas and Mateja Jamnik

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

Abstract. Heterogeneous reasoning refers to theorem proving with mixed diagrammatic and sentential languages and inference steps. We introduce a heterogeneous logic that enables a simple and flexible way to extend logics of existing general-purpose theorem provers with representations from entirely different and possibly not formalised domains. We use our heterogeneous logic in a framework that enables integrating different reasoning tools into new heterogeneous reasoning systems. Our implementation of this framework is MixR – we demonstrate its flexibility and extensibility with a few examples.

Keywords: interactive, heterogeneous, diagrammatic, theorem proving.

1 Introduction

Theorem provers generally use a sentential logical language with which they express formulae and construct proofs. However, human mathematicians typically use not only multiple but also *informal* representations such as diagrams or images within the same problem for different parts of the solution. Often, there may be parts of the problem that can be better (more intuitively or more efficiently) solved in one representation, language or theorem prover, and other parts in another. However, existing general-purpose theorem provers currently use sentential logics only and do not provide support for diagrammatic and heterogeneous reasoning. Moreover, tools for combining systems (e.g., OpenBox [1], Sledgehammer [2], Omega [6], HETS [9], Chiron [4]) do not allow augmentations to theorem provers that would enable flexible and heterogeneous mixing of formal as well as informal representations and reasoning steps within the same proof attempt. They also do not have the ability to integrate foreign or non formalised data into formulae of theorem provers, and reason with it natively. We analyse related work in more detail in Sec. 5.

Our goal is to enable *heterogeneous reasoning* (**HR**) [11], that is, reasoning with mixed representations and also with inference steps from different existing *sentential reasoners* (**SR**) as well as *diagrammatic reasoners* (**DR**). Furthermore, when logical formalisation of a particular representation (e.g., diagrams, images, or audio) is not tractable, we want to allow the embedding of such data in existing provers and still enable informal heterogeneous reasoning with these opaque objects within an otherwise formal proof. Our aim is to provide an HR

T. Dwyer et al. (Eds.): Diagrams 2014, LNAI 8578, pp. 277–292, 2014.

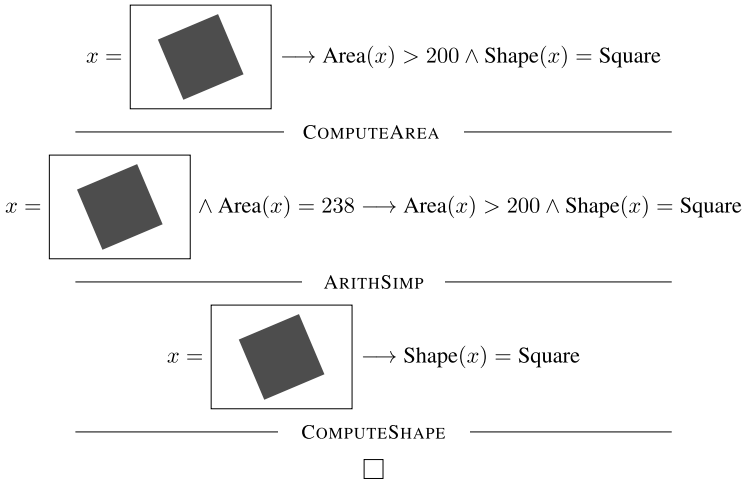

Fig. 1. A heterogeneous proof: it consists of three proof steps. The ComputeArea inference step is heterogeneous. It takes a bitmap image and extracts some information (the area of the square) which is expressed in the sentential language. The ArithSimp inference step is sentential. The ComputeShape is also a heterogeneous inference step. It extracts that the bitmap shape is a square and thus resolves the implication.

framework that enables the construction of heterogeneous proofs. Fig. 1 shows a simple example of a heterogeneous proof that we want to construct in an HR system. The idea is that the user can choose the most appropriate sentential and diagrammatic reasoners and representations, integrate them using our framework, and produce proofs where they can readily choose which parts of the proof they want to represent with which language, and which parts they want to prove with which reasoner.

Our work aims to model human flexible and informal reasoning with their plethora of representations and reasoning techniques. The applications of our work are targeted at tool developers whose sentential or diagrammatic theorem provers' power could be enhanced by bringing to them new and possibly informal representations and reasoning tools. Moreover, domain specific tools like, for example, those for image processing, circuit design, natural language processing, Venn and spider diagrams, which typically do not have access to reasoning engines, can utilize our framework to gain formal reasoning capabilities. Thus, the main contributions of our work are:

- A generic infrastructure for extending existing general-purpose theorem provers (**TP**) with heterogeneous reasoning – we call this *heterogeneous logic* and describe it in more detail in Sec. 2;
- A mechanism, called *placeholders*, for embedding foreign data into formulae of existing theorem provers: it is a crucial part of our heterogeneous logic;
- The MixR framework, which is an implementation of the heterogeneous logic and placeholders. It is a generic infrastructure for creating new HR systems.

MixR can integrate arbitrary existing TPs of any modality with each other into new HR systems – we describe the architecture of MixR in Sec. 3.

A tool developer can plug their chosen reasoners into MixR by writing MixR drivers for them. MixR, in turn, integrates them with each other into a new HR system. For example, we plugged Speedith [13] for spider diagrams and Isabelle [10] for sentential higher-order logic into MixR to create the Diabelli [12] HR system (see Sec. 4.1). MixR provides a user interface as well as an application programming interface (**API**) for drivers. Using the API, the drivers can share, translate and visualise formulae of various modalities. They may also apply foreign inference steps and query other drivers to invoke foreign reasoning tools.

MixR provides placeholders which store foreign data that is dealt with by external tools. This data is directly embedded into formulae of a prover which treats them as primitive objects that can be reasoned with its standard inference engine. When required, the reasoner can invoke external tools on this data to obtain new knowledge as the result. Our approach using placeholders removes the need for translations between representations which is particularly useful when no such translation is available or even possible (e.g., diagrammatic representations from CAD tools, images, and signal processing).

We demonstrate the generality and extensibility of MixR in Sec. 4 by presenting three examples: Diabelli for mixing spider diagrams and sentential higher-order logic, and two new prototype drivers: one for image processing, and another for natural languages. These show how to integrate tools and languages of different modalities with existing TPs. We also show that HR is achievable by merely extending general purpose TPs rather than creating entirely new ones.

2 Heterogeneous Logic

Our heterogeneous logic provides a generic infrastructure to formally or informally connect multiple logics as well as representations with each other. It serves as the foundation for building HR frameworks – MixR is its example implementation (see Sec. 3). We first define the basic concepts and then the logic of HR.

■ **Participating logics and reasoners** are integrated by our heterogeneous logic into a single system. Participating logics may either be diagrammatic, sentential, or of another modality. Participating reasoners provide languages, inference rules, theories, and proof structure. Goal-providing reasoners are *master* reasoners, and others are *slave* reasoners. Master reasoners must provide a language, proof obligations (goals) expressed in that language, inference steps, and a concept of a proof. Slave reasoners must only provide inference rules.

Reasoners may be formal and logical (e.g., a sentential general-purpose theorem prover, a formal diagrammatic prover), or informal (e.g., CAD software, image processing, or signal processing tools whose procedures for knowledge extraction have not been formally verified).

■ **Participating languages and inference rules** are provided by participating reasoners and can be diagrammatic, sentential, or of other modality. We denote the *set of all participating languages* with $\mathbb{L}$, and the *set of all participating*

inference rules with $\mathbb{I}$. Our heterogeneous logic does not impose any restrictions on the syntax and semantic interpretations of these languages – they are left for the reasoner to provide. The languages can be formal or informal. For example, general-purpose TPs typically use formal logical languages and inference rules. Image processing software, however, uses informal images and informal image processing algorithms as its inference rules for extracting knowledge (e.g., a trend in a chart). Thus, the languages express and store data, while the inference rules (algorithms) extract knowledge.

■ **Heterogeneous formula** is a family of representations of the same original expression but possibly in different languages. A single representation is denoted with ϕ_a and is expressed in a particular language $\mathcal{L}_a \in \mathbb{L}$.

Definition 1. *A* ***heterogeneous formula*** *is a pair* (ϕ_m, R)*, where* ϕ_m *is the main representation and* R *is a set of derived representations:* $R = \{\phi_a,\ \phi_b,\ \ldots\}$.

Every heterogeneous formula has a specific *main* representation ϕ_m from which others can be derived. These derived representations in the set R are in some semantic relation to the main representation ϕ_m which must be specified either by the translation procedure or by the inference rule that produced the representation. Our heterogeneous logic does not impose any restrictions on how these translations or inferences are defined (see below).

The languages of representations may be sentential, diagrammatic, or can contain multiple embedded foreign expressions. Representations must be assertions that can be evaluated to establish their truthfulness. The reasoner that provides the language defines and manages this notion of truth. Some languages (such as formal logics) have a clear definition of truth, while others (such as natural languages) lack formality. Nonetheless, many of these informal domains have notions of assertions that are compatible with the ones in formal TPs and can still be partly defined. We exploit this via placeholders, described below.

■ **Heterogeneous goal**, Γ, is a heterogeneous formula consisting of *premises* (heterogeneous formulae) and a *conclusion* (a single heterogeneous formula):

$$\Gamma = P_1, P_2, \ldots, P_n \Longrightarrow C$$

Each premise and the conclusion may be represented in a different language. Goals provide a standard way to exchange proof obligations and proofs between participating reasoners. Each reasoner can be invoked on parts of the goal which can be (re)presented in its language. Goals originate from a master reasoner. If they are expressed in one language, then they are homogeneous. If they are expressed in different languages or contain placeholders for non-formalisable or non-translatable languages, then they are heterogeneous.

■ **Placeholder** (or embedded foreign formula) is our novel concept that allows sentences of one language to be inserted directly (without translation) into a host formula of another language. Placeholders support embedding of representations that are foreign to a participating language (e.g., in Fig. 1 an image is embedded within a sentential logical formula). This is particularly useful when a translation of one, possibly informal, representation into another is not available or even

possible. The novelty of placeholders is that they do not require translation or changing the syntax of the participating language. Instead, placeholders *encode* the foreign formula using the syntax of the host formula (for a concrete example of a placeholder, see Sec. 4.2). This differs from existing work on logical theories since they do not admit foreign data, they only use uninterpreted constants for building theories in their own, *single* language. We, in contrast, use uninterpreted constants to carry foreign data expressed in any language. This data becomes part of a chosen host language and theories can be built using it.

Definition 2. *A* ***placeholder*** $\pi[\mathcal{V}, \mathcal{L}_p, \phi_p]$ *is a formula in the language* $\mathcal{L}_c \in \mathbb{L}$*, where* ϕ_p *is a* payload formula*,* $\mathcal{V}$ *is a set of variables bound implicitly in* ϕ_p*, and* $\mathcal{L}_p \in \mathbb{L}$ *is the language in which* ϕ_p *is expressed.*

The role of the placeholder $\pi[\mathcal{V}, \mathcal{L}_p, \phi_p]$ *in a sentence of the language* $\mathcal{L}_c$ *is the same as that of a Boolean predicate* $P(v_1, \cdots, v_n)$ *where* $\{v_i \mid 1 \leq i \leq n\} = \mathcal{V}$*.*

Variables $\mathcal{V}$ provide the link between the host and foreign formulae. For example: using an external reasoner, we can deduce that for any x and any y the placeholder $\pi[\{x, y\}, \mathcal{L}_{nl}, \text{“x is smaller than y”}]$ implies that $x < y$ in the host theory. Note that universal quantification cannot be to the right of the implication, since x and y are bound in the placeholder. If no variables were listed in $\mathcal{V}$, for example, $\pi[\emptyset, \mathcal{L}_{nl}, \text{“x is smaller than y”}]$, then x and y are independent universally quantified free variables in the host theory, and now the placeholder implies that $x < y$ for any x and any y, a clear falsehood.

Placeholders have no defined interpretation in the host reasoner. They only carry data (i.e., the payload) throughout a proof while being embedded within a host formula. The data is given to external reasoners for knowledge extraction. The extracted knowledge is then embedded back into the proof hosted by the master theorem prover. The external reasoners can themselves be formal, but the entire step is still informal due to placeholder's informal embedding of data.

■ **Inference rules** $I \in \mathbb{I}$ are applied on goals to constitute inference steps. In general, an inference step takes as input a set of initial HR goals $\{\Gamma_1, \ldots, \Gamma_n\}$ and produces a new, transformed set of $\{\Gamma'_1, \ldots, \Gamma'_m\}$. The inference rule I must guarantee that the new goals *logically entail* the initial goals (see below).

The general form of an inference step in our heterogeneous logic is:

$$\frac{\Gamma'_1, \ldots, \Gamma'_m}{\Gamma_1, \ldots, \Gamma_n} \text{ rule: } I.$$

We use the backward reasoning notation, where transformed goals are on top and the original ones on the bottom. Inference rules may transform premises as well as the conclusion of a goal, thus both forward and backward (goal-directed) reasoning methods are supported in our heterogeneous logic.

Inference rules adhere to typical constraints of goal-directed reasoning: strengthening or information-preserving rules can be applied on the conclusion of a goal; weakening or information-preserving rules can be applied on the premises of a

goal. Our heterogeneous logic defines weakening, strengthening, and information-preservation relative to other languages on the basis of translations.

Consistency: Our heterogeneous logic does not impose any restrictions on how the inference rules are constructed. Consequently, no guarantees can be given about the consistency and correctness of proofs in general (see *heterogeneous proof* below). Nonetheless, our logic can formally guarantee correctness of proofs under the following conditions: the entire proof must be hosted in a single master reasoner, the proof must start in the language of the master reasoner, all inferences (including external ones) must produce goals in the language of the master reasoner or the produced goals can be translated into that language. Under these conditions, the master reasoner can verify the correctness of each step (this process is called proof reconstruction). On the other hand, proofs that extract knowledge from placeholders are necessarily informal, as no translation of their content into the language of a master reasoner exists.

Heterogeneity: An inference rule is *heterogeneous* if it transforms goals containing formulae of different languages, otherwise it is *homogeneous*. Heterogeneous rules have to be written by the developer of a heterogeneous system (e.g., heterogeneous inference rules in Sec. 4.3 take an image and produce a sentential formula). Heterogeneity is also achieved by *translating* formulae of different languages to a target language before passing them to the inference rule.

■ **Translation procedure** takes an existing formula representation of a premise, conclusion, or entire goal, and produces another one in another language. These representations must be in a correct logical entailment relation with each other. Depending on the entailment of translation we distinguish between translations that are: *strengthening* (applicable on conclusions), *weakening* (applicable on premises) and *information-preserving* (applicable anywhere). If entire goals are translated, weakening applications must not be used.

Consistency and validity: Translations are formal when the logic of one language, say $\mathcal{L}_a$, has been formalised in the logic of another language, say $\mathcal{L}_b$. If these translations are used in a proof, then such a proof is formal if hosted and reconstructed by the master reasoner that uses the language $\mathcal{L}_b$. Our formalisation of spider diagrams in the sentential logic of Isabelle/HOL is one such formal translation: proofs are hosted and reconstructed in Isabelle/HOL [12]. Translations can be left informal when formalisation is not feasible. Clearly, in this case their soundness cannot be guaranteed.

■ **Logical entailment** between two formulae is *heterogeneous* if the two formulae are of different languages, and *homogeneous* otherwise. The heterogeneous logical entailment is defined for formulae resulting from applying inference steps or translations procedures as:

Definition 3. *Formula ϕ_a, expressed in language $\mathcal{L}_a \in \mathbb{L}$,* **entails** *formula ϕ_b of language $\mathcal{L}_b \in \mathbb{L}$ with respect to language $\mathcal{L}_c$ if there exists a direct translation t_1 from ϕ_a to ϕ_a' and another translation t_2 from ϕ_b to ϕ_b', where both ϕ_a' and ϕ_b' are expressed in language $\mathcal{L}_c$, and ϕ_b' can be deduced from ϕ_a' in a finite homogeneous proof, or if for any interpretation such that ϕ_a' is true, so is ϕ_b'.*

Heterogeneous entailment is formal with respect to language $\mathcal{L}_c$ if the translations t_1 and t_2 are formalised within the logic of $\mathcal{L}_c$ (e.g., our formalisation of spider diagrams in Isabelle/HOL in [12]). Homogeneous entailment is a special case of heterogeneous entailment, where the translation is an identity map.

■ **Heterogeneous proofs** are based on discharging proof obligations which take the form of heterogeneous goals. An HR proof thus consists of initial heterogeneous goals, followed by multiple applications of *heterogeneous inference rules* which produce new goals. The inference rules are applied until only tautological goals remain – they are thus discharged and the proof is completed.

Soundness: Proofs in our heterogeneous logic are hosted in a master reasoner. Foreign inference rules can be used on goals by extracting the goals from the master reasoner, passing them to the slave reasoner for inference, and then inserting the result back into the master reasoner's proof. Only a formal translation can provide soundness and completeness guarantees. So, if the new formula is *not* a placeholder (i.e., a translation into the host language exists), then the master reasoner can verify its soundness. Otherwise, the master reasoner can trust that the new goal is correct using the oracle principle, but clearly the proof cannot be guaranteed to be sound. Tools such as HETS [9] could be utilised to provide other existing translations, and thus formal guarantees for them.

3 Architecture and Design of MixR

MixR implements the heterogeneous logic introduced in Sec. 2: it manages formulae, their translations, inference rules and communicates these with participating reasoners. Tool developers, who are the users of the MixR framework, can integrate their own representations (formula formats), translation procedures, inference rules, visualisations, or entire participating reasoners using MixR by providing drivers which are pluggable components. This results in a new *HR system* for end users. We present three examples of drivers in Sec. 4. MixR was implemented in Java using the NetBeans platform.[1]

Fig. 2 shows the general architecture of MixR. A MixR-based system adds HR on top of participating reasoners. *Master reasoners* host the proof in their proof management infrastructure. MixR, however, takes proof obligations (goals) from master reasoners, translates them, visualises them, and lets users interact and explore them. MixR also facilitates communication of goals with other *slave reasoners*. Slave reasoners transform the goals with their inference rules. The result is communicated back to the originating master reasoner. In summary, MixR's role is that of a mediator between multiple master and slave reasoners.

MixR is a small core that implements a set of *heterogeneous logic components* (concepts from our heterogeneous logic), *driver contracts* (that facilitate integration of reasoners), and *general UI components* (which display pending proof obligations, enable user interaction and coordinate formulae visualisation).

[1] All MixR sources are available from `https://github.com/urbas/mixr`.

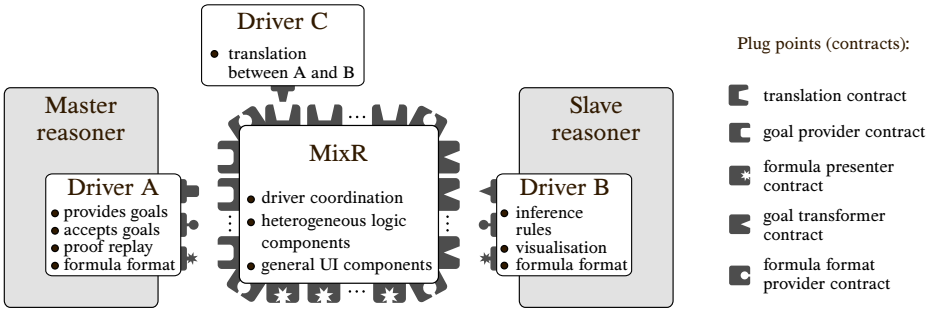

Fig. 2. The outline of MixR's architecture with hypothetical drivers. The central box represents MixR's core. It contains the implementation of heterogeneous logic components, general UI components, and driver plug-points. Drivers surround MixR's core and plug into it through the plug-points.

3.1 Heterogeneous Logic Concepts

Since heterogeneous proofs are hosted by master reasoners, it is the responsibility of master reasoners (and their drivers – see below) to implement the concept of heterogeneous proofs. MixR provides implementations for the other heterogeneous logic concepts: `FormulaFormat` (participating languages), `FormulaRepresentation`, heterogeneous `Formula`, heterogeneous `Goal`, `Placeholder`, heterogeneous `InferenceRule`, and `FormulaTranslator` (translation procedures). MixR also implements placeholders that provide the infrastructure for inserting foreign data into formulae of existing theorem provers (for examples, see Sec. 4). With these, specialised ad-hoc mixed systems do not have to repeat implementing or re-inventing placeholders. Instead, MixR allows the developer to focus on the rules and simply reuse the infrastructure.

3.2 Driver Contracts

MixR makes the functionality of every driver available to every other driver and also the user interface. To simplify and generalise all inter-driver communication, MixR defines a standard set of driver contracts: `GoalProvider`, `FormulaFormatProvider`, `FormulaPresenter`, `TranslationProvider`, and `GoalTransformer`.

The `GoalProvider` contract must be implemented by drivers that connect master reasoners to MixR. They communicate pending proof goals of the master reasoner to MixR. MixR also uses this contract to re-insert transformed goals back into the master reasoner.

Drivers providing new formula formats to MixR must implement the `FormulaFormatProvider` contract. They must provide a unique name of each format, a description and an API for manipulating the formulae of that format.

MixR gives users the choice of using sentential, diagrammatic as well as heterogeneous formulae. These are displayed to the user in a way most suitable for the given format. For this reason, drivers can implement the `FormulaPresenter` contract that facilitates the visualisation of formulae of particular formats.

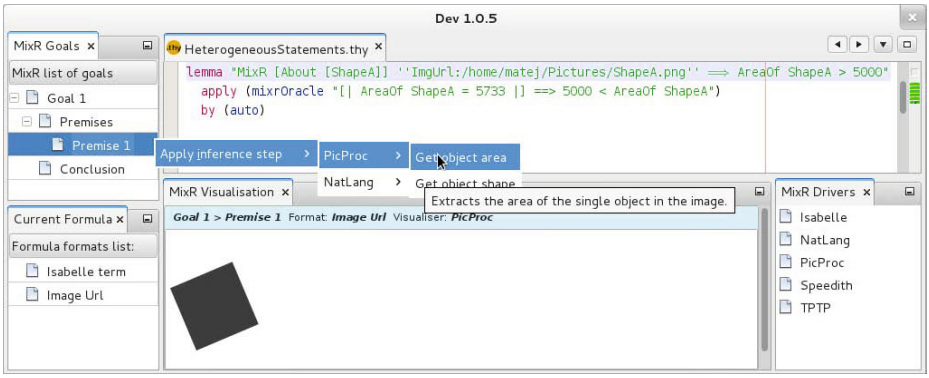

Fig. 3. The user interface of a MixR-based system that integrates PicProc with Isabelle

Translation procedures are integrated into MixR by drivers implementing the `TranslationProvider` contract. These drivers advertise all translations they contribute. MixR uses them automatically whenever a particular formula format is requested. The user may also invoke a translation on any formula interactively.

Drivers that transform goals with inference rules must implement the `GoalTransformer` contract. They bring slave reasoner functionality into MixR.

3.3 General UI Components

MixR's general UI components present and control aspects that are common to the entire framework. Fig. 3 shows the user interface of a system created using MixR, which contains the following components (sub-windows):

- Top-left: The list of pending goals of the currently active master reasoner.
- Top-right: The proof script of the master reasoner.
- Pop-up menu under the mouse cursor: Shows how the user interacts with the list of pending goals (e.g., applying inference rules on the goals, translating them to a target representation, or selecting the part to be visualised).
- Bottom-centre: Visualises currently active or selected formulae.
- Bottom-left and bottom-right: Lists of all available reasoning languages, registered drivers, master reasoners, and slave reasoners.

This is how the interaction with inferences is carried out in MixR:

1. The user first selects the part of a goal on which to apply the inference rule. In Fig. 3 the user selected the first premise of the first goal (see top-left sub-window).
2. Next, the user chooses an inference rule by left-clicking the selected invocation target and choosing the desired inference rule (shown under the mouse cursor in Fig. 3).
3. Now, if the inference rule is an *interactive* one, the user may be asked for more input. In the example from Fig. 3 the selected rule *Get object area* requires no additional input.

4. Finally, the inference rule returns its result, which may be a formula foreign to the target master reasoner. If so, MixR tries to translate it. If no translation succeeds, the result is placed into a placeholder. MixR constructs a new goal with the result and passes it to the master reasoner's driver. The driver then inserts the goal into the master reasoner's proof script. In Fig. 3, the line starting with `apply` was inserted after the user invoked the inference.

4 Prototype Examples

We now demonstrate MixR's generality and its facilities to integrate diverse systems into heterogeneous reasoning tools. To integrate a new reasoner, we have to create a driver for it. The driver must implement a particular set of driver contracts to integrate the desired features into MixR. In the past, we used MixR to integrate the general purpose theorem prover Isabelle [10] and our diagrammatic prover Speedith [13] into an HR system Diabelli [12]. Here we present Diabelli, and two new drivers: *NatLang* and *PicProc*. *NatLang* integrates a mock sentential reasoner for informal natural languages. *PicProc* integrates an image processing visual reasoning tool. We use MixR to combine each with Isabelle that has the role of the master reasoner in this demonstration. These prototype drivers work on domains that are inherently hard to formalise. They showcase how such informal domains may still be integrated with a formal general purpose theorem prover. To give an idea of their complexity, the Speedith, Isabelle, NatLang and PicProc drivers each consist of between 400 and 600 lines of Java source code.

4.1 Diabelli

Diabelli is our HR system that allows users to construct heterogeneous proofs with spider diagrams mixed with the sentential higher-order logic language. It is a *system* resulting from plugging the Speedith [13] DR and the Isabelle [10] SR into the MixR *framework*. Diabelli does not utilise MixR's placeholder mechanism since Speedith's representation can be translated into Isabelle's, so Speedith's data is not foreign to Isabelle. Fig. 4 shows Diabelli's user interface with Isabelle/HOL sentential formulae mixed with Speedith's spider diagrams and spider-diagrammatic inference rules. The implementation details of the drivers for Speedith and Isabelle can be found in [12]. Diabelli provides a non-hypothetical, fully integrated, and non-trivial experimental evidence for how MixR contributes and aims to spur innovation within the community using and studying diagrams.

4.2 NatLang

The NatLang driver implements the contracts `FormulaFormatProvider` and `GoalTransformer`. It provides a single formula format, the `NatLangFormat`, which is a plain string with no restrictions or grammar. We use MixR's

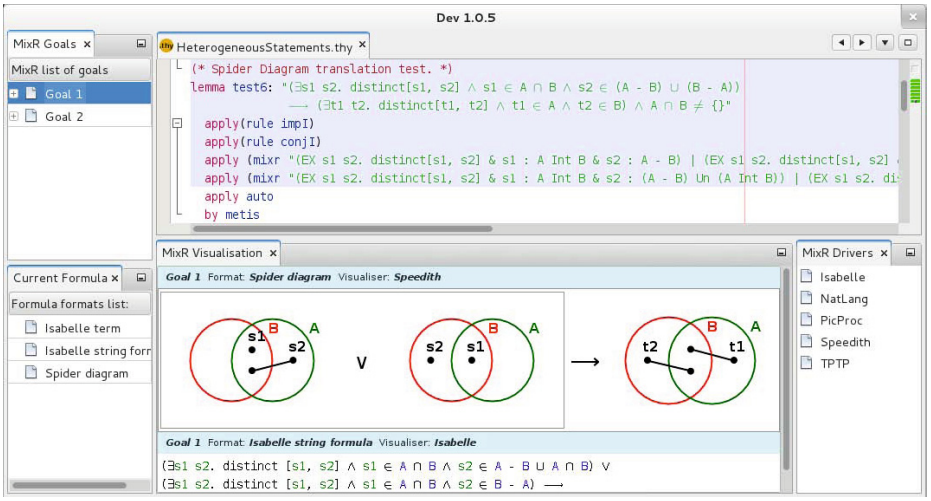

Fig. 4. Diabelli's main window. The top-right sub-window contains I3P's hosting proof script editor for Isabelle. Users may edit theories in the same way as in standalone I3P. Diabelli inserts heterogeneous proof steps into this proof script as the user applies them through Diabelli's GUI (by right-clicking on the goals in the top-left sub-window).

placeholders to insert NatLang expressions into Isabelle/HOL formulae. Here is an Isabelle/HOL formula containing one such placeholder:

```
(MixR [About[Teenager, Human]] "NatLang:Every teenager is a
    human.")
      ∧ (∀p. p ∈ Human ⟶ p ∈ Mortal)
⟶ Teenager ⊆ Mortal.
```

The predicate `MixR` is the placeholder. It contains the set of variables ($\mathcal{V} = \{Teenager, Human\}$) and a string (*payload*) which consists of the name of the foreign language ($\mathcal{L}_p = NatLang$) and the actual foreign formula ($\phi_p = Every\ teenager\ is\ a\ human$).

NatLang provides inference rules `HomogeneousInference` and `HeterogeneousInference`, which rely entirely on user-input. The homogeneous inference rule takes a NatLang expression and returns a NatLang formula which is re-embedded into the Isabelle/HOL formula as a placeholder. The heterogeneous rule takes a NatLang expression and returns an Isabelle/HOL formula, which it places into the Isabelle formula without a placeholder. Here is an example where $Teenager \subseteq Human$ has been inferred from the above NatLang sentence:

```
(Teenager ⊆ Human ∧ (∀p. p ∈ Human ⟶ p ∈ Mortal)) ⟶
    Teenager ⊆ Mortal
```

The resulting goal may now be reasoned about with Isabelle's own theories and inferences. The homogeneous inference rule shows how a foreign inference step can be used within an Isabelle proof (i.e., both the input and the output formulae are foreign to the theorem prover). The heterogeneous inference rule, on the other hand, demonstrates how knowledge can be transferred between

the natural language domain and the theorem prover's domain. Such reasoning is informal as a whole, but the informal parts are limited to small steps that link both domains. This mechanism provides more than the oracle principle: MixR extracts knowledge from the placeholder's data piecewise (as it's needed in the proof); the oracle principle is used to trust each such step of knowledge extraction, translation, and re-embedding.

4.3 PicProc

The PicProc example demonstrates the integration of images and image processing into sentential theorem provers. This example also utilises placeholders and the Isabelle master reasoner. Fig. 1 shows a heterogeneous proof that PicProc contributes to, and Fig. 3 shows the screenshot of the system resulting from plugging PicProc and Isabelle into MixR.

Similarly to the NatLang driver, PicProc provides a new formula format and two inference rules. The formula format is called `ImgUrl` and the provided inferences are `ComputeArea` and `ComputeShape`. The ImgUrl formulae are paths to bitmap images. The rule `ComputeArea` returns the area of the shape (the tilted square at the bottom of Fig. 3). The rule `ComputeShape` infers the type of the shape. This rule decides whether the shape is a circle, a triangle, or a square (based on the ratio between the area and the circumference of the shape).

Note that in Fig. 3 (top-right sub-window) the PicProc image is inserted into a `MixR` placeholder within the host Isabelle/HOL formula. The `About[ShapeA]` clause indicates that the placeholder talks about an object called `ShapeA`. Placeholder's variables $\mathcal{V}$ such as `ShapeA` must be extracted and listed by the driver (in this case the driver simply takes the name of the image file), otherwise Isabelle cannot know that the payload contains knowledge about them.

The PicProc driver also implements the `FormulaPresenter` contract which provides MixR with new visualisation capabilities. ImgUrl formulae found by MixR can now be passed to PicProc, which opens the image and displays it in MixR's visualisation panel (see the bottom-centre sub-window of Fig. 3).

5 Related Work and Evaluation

The topic of HR has been explored in a number of different contexts [11,1,12]. Here, we compare our work to existing logics and systems that provide some level of heterogeneity. We then evaluate its scalability and extensibility.

5.1 Logics

Chiron [4], context logic [3], and multilanguage hierarchical logics (MLHL) [5] are logics that provide heterogeneity on two levels: multiple reasoning paradigms with a single sentential language (Chiron and context logic), or single reasoning paradigm with multiple first-order sentential languages. In particular, these logics use purely sentential languages and establish entirely formal connections

between all their components. For example, Chiron uses a *single* sentential language but allows switching between different reasoning paradigms (e.g., classical reasoning, set-theoretic reasoning, type-theoretic reasoning). Similarly, context logic uses a *single* sentential language, but its semantics are defined with contexts and multiple sets of statements. This enables context logic to encompass intuitionistic, classical, and predicate logic as its special cases. The goal of MLHL is similar – to enable reasoning in multiple sentential logics. Although MLHL allows multiple different languages, they are all confined to first-order logics.

Thus, Chiron, context logic, and MLHL have a different notion of heterogeneity to ours, and unlike MixR they also do not extend logics of existing theorem provers with HR, nor facilitate *formal and informal* HR.

5.2 Frameworks and Systems

■ **OpenBox** is the only other existing implementation of an HR framework [1]. It maintains its own proofs and uses external theorem provers to validate separate inference steps within its proofs. These provers may be diagrammatic or sentential, which makes OpenBox a heterogeneous framework. Unlike MixR, OpenBox does not extend existing reasoners with heterogeneous reasoning but rather utilises them to make its own proofs heterogeneous. This forces users of reasoners to abandon their existing work. In contrast, MixR reuses proofs and proof scripts of existing reasoners which means that users can retain their existing work and seamlessly extend it with HR. While separate formulae of OpenBox's proof can be of different modalities, a single OpenBox formula is of a single modality. This is in contrast to MixR where a single formula can contain embedded foreign formulae of different modalities. Furthermore, unlike MixR, OpenBox cannot embed foreign data into formulae of existing TPs.

■ **Sledgehammer** [2] is a component of the Isabelle [10] SR that passes translations of Isabelle statements to external reasoners for validation. It replaces external answers with corresponding Isabelle's inference steps. This purely sentential process requires formal translations and that obtained answers are fully expressible with Isabelle. Sledgehammer cannot process heterogeneous nor informal inference steps, and it cannot embedding foreign data into formulae.

■ **Omega** is a proof planning system [6] that uses different sentential reasoners, and is thus a homogeneous rather than a heterogeneous system. Unlike MixR, Omega imposes its own proof structure that is maintained by the blackboard mechanism. An external SR can only be invoked if the translation of the Omega formula into its representation exists. This is in contrast to MixR which can use placeholders to embed foreign data from external reasoners into formulae when no translation exists. Omega also does not extend existing reasoners in any way.

■ **HETS** (or *Heterogeneous Tool Set*) [9] differs from MixR in the use of the term "heterogeneous". In HETS it is used to refer to formal relations between multiple sentential logics. Unlike MixR, HETS is thus a purely sentential system, which produces formal translations between sentential logics. In contrast to MixR, which does not require translations, a logic cannot be used in HETS if it cannot be translated into other HETS logics.

5.3 Scalability and Extensibility

Current benchmark problem sets (such as TPTP) test the efficiency and scope of theorem provers in first-order or higher-order sentential logic. In contrast, our heterogeneous logic and framework aim to expand the vocabulary of existing TPs with foreign data, formulae, diagrams, informal reasoning, and foreign inference – none of these are included in benchmark sets. Thus, rather than quantitatively we qualitatively evaluate the extensibility of our heterogeneous logic and framework.

The *range of domains* that can be integrated using MixR was exemplified in Sec. 4 where we presented case-studies of Diabelli, NatLang and PicProc. These embed spider diagrams, natural language and image processing into an existing theorem prover. We now evaluate the *generality* of our framework by assessing if and which other reasoners can be plugged into it. We consider Cinderella [8], Hyperproof's Blocksworld [1] and Diamond [7] as representative DRs, and HOL4, HOL Light, Coq and Twelf as representative SRs.

Cinderella [8] (a reasoner for geometric constructions) and Diamond [7] (a reasoner for diagrammatic proofs on natural numbers) could both be master reasoners in MixR. Their drivers would have to implement UI editors for hosting diagrammatic proof scripts and visualisation procedures. Both, Cinderella and Diamond use automated sentential TPs to validate and verify their diagrammatic proof steps. These TPs could be integrated as slave reasoners in MixR. Cinderella could also be used to integrate the domain of geometry into other master reasoners such as Isabelle. This would require translation procedures between Cinderella's internal representation and Isabelle/HOL formulae.

Blocksworld expresses relations between 3D objects placed on a checkerboard. The reasoning about these relations is done within Hyperproof [1] using a first-order logic TP. Thus Blocksworld does not have a notion of a proof state, and can only be plugged into MixR as a slave reasoner. Its driver would have to integrate its visualisation, and also implement a translation procedure between Blocksworld's models and any existing master reasoner in MixR.

Interactive theorem provers HOL4, HOL Light, Coq, and Twelf all use textual proof scripts. Master reasoners need these to host proofs. All these provers enable reasoning with either oracles or axioms (required for informal reasoning). HOL4 and HOL Light provide the `mk_thm` command to introduce informal inferences; Coq provides the command `Axiom`; and Twelf supports axiomatic inference rules as part of its meta-theorem infrastructure. MixR's placeholders require uninterpreted constants and functions in the theorem prover's logic. HOL4 and HOL Light support uninterpreted functions through the `def_constant` command. Similarly, Coq provides the command `Variable` for this purpose. Finally, Twelf's fundamental approach to building theories is to start with uninterpreted symbols and provide inference rules for them later. Therefore HOL4, HOL Light, Coq, and Twelf could all be extended with our heterogeneous logic and thus plugged into MixR. This would require the developers to write additional translation procedures, communication channels between the theorem prover and MixR, and the integration of the theorem prover's text editor software with MixR. Theorem provers that are fully automated, such as Z3, ACL2, Vampire, and Spass lack

proof scripts (or other theory- and proof-specification documents). Thus they can be integrated into MixR as slave reasoners but not as master reasoners.

6 Conclusion

In this paper, we presented a heterogeneous logic that marries formal logics with diverse and possibly informal representations. Our novel concept of placeholders enables existing logics to formally treat foreign representations within their own formulae. We implemented this logic in a heterogeneous framework MixR that facilitates a flexible integration of sentential or diagrammatic theorem provers and other formal and informal reasoners, representations and visualisations. These can be plugged into MixR via drivers to produce integrated HR systems. With MixR, we can explore in breadth and in depth the interaction between diagrammatic languages and formal sentential languages which invites multi-domain collaboration and exploration. Domains traditionally inaccessible to formal reasoners, can now be integrated and exploited. The developers of HR systems can be flexible about their system design: their choices may depend on issues surrounding efficiency, intuitiveness of proofs, or level of expertise of end users of resulting systems.

We presented three examples of integrated systems from diverse domains of spider diagrams, natural language and image processing. Plugging them into MixR alongside a general purpose theorem prover enabled for the first time to formally use the information from informal reasoners to construct proofs.

Many reasoning tools, representations and visualisation aids in AI exist mostly in isolation, specialised in their specific domains. Bringing them together in a simple, flexible and formal way allows them to contribute to the problem solving/theorem proving tasks. We believe this is desirable for several reasons: it better models what people do in problem solving, it allows developers to easily design systems that are flexible according to the needs of the end-users, and it enables us to take advantage of the existing powerful technology out there in a novel and sustainable way.

Acknowledgements. This work was supported by EPSRC Doctoral Training Grant and Computer Laboratory Premium Research Studentship (Urbas).

References

1. 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)
2. Böhme, S., Nipkow, T.: Sledgehammer: Judgement day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS (LNAI), vol. 6173, pp. 107–121. Springer, Heidelberg (2010)
3. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic as modal logic: completeness and parametric inexpressivity. In: POPL, pp. 123–134. ACM (2007)

4. Farmer, W.M.: Biform Theories in Chiron. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 66–79. Springer, Heidelberg (2007)
5. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics, or: How we can do without modal logics. AI 65(1), 29–70 (1994)
6. Siekmann, J.H., et al.: Proof development with ΩMEGA. In: Voronkov, A. (ed.) CADE-18. LNCS (LNAI), vol. 2392, pp. 144–149. Springer, Heidelberg (2002)
7. Jamnik, M., Bundy, A., Green, I.: On Automating Diagrammatic Proofs of Arithmetic Arguments. JOLLI 8(3), 297–321 (1999)
8. Kortenkamp, U., Richter-Gebert, J.: Using automatic theorem proving to improve the usability of geometry software. In: Procedings of the Mathematical User-Interfaces Workshop, pp. 1–12 (2004)
9. Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set, Hets. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)
10. Paulson, L.C.: Isabelle - A Generic Theorem Prover. LNCS, vol. 828. Springer, Heidelberg (1994)
11. Shin, S.J.: Heterogeneous Reasoning and its Logic. BSL 10(1), 86–106 (2004)
12. Urbas, M., Jamnik, M.: Diabelli: A heterogeneous proof system. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 559–566. Springer, Heidelberg (2012)
13. Urbas, M., Jamnik, M., Stapleton, G., Flower, J.: Speedith: A diagrammatic reasoner for spider diagrams. In: Cox, P., Plimmer, B., Rodgers, P. (eds.) Diagrams 2012. LNCS (LNAI), vol. 7352, pp. 163–177. Springer, Heidelberg (2012)