Labelled Unit Superposition Calculi for Instantiation-based Reasoning

Konstantin Korovin* and Christoph Sticksel

School of Computer Science
The University of Manchester
korovin@cs.man.ac.uk, sticksel@cs.man.ac.uk

Abstract. The Inst-Gen-Eq method is an instantiation-based calculus which is complete for first-order clause logic modulo equality. Its distinctive feature is that it combines first-order reasoning with efficient ground satisfiability checking which is delegated in a modular way to any state-of-the-art ground SMT solver. The first-order reasoning modulo equality employs a superposition-style calculus which generates the instances needed by the ground solver to refine a model of a ground abstraction or to witness unsatisfiability.
In this paper we present and compare different labelling mechanisms in the unit superposition calculus that facilitates finding the necessary instances. We demonstrate and evaluate how different label structures such as sets, AND/OR trees and OBDDs affect the interplay between the proof procedure and blocking mechanisms for redundancy elimination.

1 Introduction

Instantiation-based methods (IMs) are a class of calculi for first-order clausal logic. The common idea is to instantiate clauses in a smart way and to employ efficient propositional or more general ground reasoning methods in order to prove unsatisfiability or to find a model. There is considerable current interest in instantiation-based methods (see [1, 9, 6]), motivated by their attractive features, some of which are complementary to other contemporary methods. Among other important properties, all known IMs naturally decide the first-order logic fragment of effectively propositional logic (EPR), also called Bernays-Schönfinkel class, which has applications in areas as diverse as bounded model checking ([11]), logic programming ([3]) and knowledge representation ([5]).

In many applications efficient equational reasoning is indispensable. While theoretical foundations of equational reasoning in several instantiation-based methods have been laid already some time ago ([8, 4, 2]), due to a number of challenging issues only now practical implementations of the calculi appear.

In this paper we take up equational reasoning in the Inst-Gen-Eq calculus as presented in [4]. One of the distinctive features of the Inst-Gen method is a modular combination of ground reasoning based on any off-the-shelf SMT solver with superposition style first-order reasoning. A major practical challenge in this approach is efficient extraction of relevant substitutions from superposition proofs, which are used for instance

* Supported by the Royal Society.

generation. Here we need to explore potentially all non-redundant superposition proofs of the contradiction, extract relevant substitutions and efficiently propagate redundancy elimination from instantiation into superposition derivations.

In this paper we address this challenge by introducing a labelled unit superposition calculus. Labels are used to collect relevant substitutions during superposition derivations and facilitate efficient instance generation. Non-trivial issues arise when we merge several superposition derivations which is done by a new merging rule. Merging allows to share several derivations of the same literal which avoids repeated work and can be used to strengthen redundancy elimination.

We introduce and investigate different label structures based on sets, AND/OR trees and OBDDs and highlight how the label structure can be exploited for redundancy elimination. Finally we have implemented these approaches and compared their performance on the TPTP library.

2 The Inst-Gen Method Modulo Equality

We consider first-order clausal logic with equality. Given a set of first-order clauses S we first form its ground abstraction $S\bot$ by mapping all variables to the same ground term, conventionally denoted $\bot$. Overloading the notation, we use $\bot$ also for the substitution that maps all variables to the ground term $\bot$. If the ground abstraction $S\bot$ is unsatisfiable (modulo equality), the original set S is also unsatisfiable and the procedure can terminate. Otherwise, there is a model $I_\bot$ of the ground abstraction $S\bot$ and the first-order instantiation process is guided by means of a selection function sel based on $I_\bot$. The selection function assigns to each first-order clause C in S exactly one literal $\mathrm{sel}(C) = L$ from C such that $I_\bot \models L\bot$. At least one such literal always exists as the ground abstraction of the clause is true in the model $I_\bot$.

If the set of selected (not necessarily ground) literals, seen as unit clauses, is satisfiable in first-order logic with equality, a model for the clause set S exists and it has thus been proved satisfiable. Otherwise, there is a subset of the selected literals which is inconsistent. We instantiate the clauses these literals are selected in such that the inconsistency can already be witnessed in the ground abstraction. Thus the ground model has to be refined in order to resolve the inconsistency. For non-equational literals it suffices to search for unifiable complementary literal pairs. In the presence of equations, we apply the unit superposition calculus in order to find inconsistent literals and to obtain instantiating substitutions.

Definition 1 (Unit Superposition).

$$\frac{l \simeq r \qquad s[l'] \simeq t}{(s[r] \simeq t)\sigma}\ (\sigma) \qquad \frac{l \simeq r \qquad s[l'] \not\simeq t}{(s[r] \not\simeq t)\sigma}\ (\sigma) \qquad \frac{l \not\simeq r}{\Box}\ (\sigma)$$

(i) $\sigma = \mathrm{mgu}(l, l')$*, (ii)* l' *is not a variable, (iii)* $l\sigma\rho \succ r\sigma\rho$ *and (iv)* $s[l']\sigma\rho \succ t\sigma\rho$ *for some grounding substitution* ρ $\qquad \sigma = \mathrm{mgu}(l, r)$

The unit superposition calculus is similar to the standard superposition calculus, see, e.g., [10]. For simplicity and without loss of generality we assume pure equational

logic where all atoms are equations. Different literals are made variable-disjoint and as we only have literals, i.e. unit clauses, we reduce the inference rules to the ones above.

A *proof* of the contradiction, denoted $\Box$, is a tree where each leaf is a selected literal in a clause, inner nodes are obtained by applying inference rules to the parent nodes and the root is the contradiction $\Box$. Every proof of a contradiction shows that the literals at the leaves are inconsistent in first-order and the clauses these literals are selected in have to be instantiated in a way that the ground solver can witness the inconsistency. Then the model of the ground abstraction has to be refined, possibly leading to further conflicts or making the ground abstraction unsatisfiable. If no contradiction can be proved from the set of selected literals, the first-order clause set is satisfiable. Let us note that unlike in standard superposition where one proof of the contradiction suffices, we need to explore all non-redundant proofs by unit superposition. Different proofs generate different clause instances and potentially all of them are necessary to witness unsatisfiability of the given set of clauses.

If in the unit superposition calculus the premises $l \simeq r$ and $s[l'] \simeq t$ can be inferred to $(s[r] \simeq t)\sigma$, then the ground abstraction $(s[r] \simeq t)\sigma\bot$ of the conclusion follows (modulo equality) from the ground abstractions of the premises instantiated with the mgu σ, namely $(l \simeq r)\sigma\bot$ and $(s[l'] \simeq t)\sigma\bot$. The same argument holds for the disequation $s[l'] \not\simeq t$. Further, if $l \not\simeq r$ can be inferred to the contradiction $\Box$ using σ, then the ground abstraction $(l \not\simeq r)\sigma\bot$ is also contradictory. By induction over the inferences in a proof of the contradiction we can show that there are *relevant substitutions* to the leaf literals such that the ground abstractions of the instantiated literals are contradictory.

The following extraction of relevant instances from proofs of the contradiction has been described and proved complete in [4]. For each leaf literal in a proof its relevant substitution is obtained by composing the substitutions from inferences along the branch. The set of *relevant instances* of a unit superposition proof is $\{C_1\theta_1, \ldots, C_n\theta_n\}$ where $\theta_1, \ldots, \theta_n$ are the relevant substitutions to the leaf literals and $C_1, \ldots, C_n$ are clauses the leaf literals are selected in. In order to witness the inconsistency in the ground abstraction and to force a refinement of the ground model, we add the relevant instances to the original clause set and form their ground abstractions.

However, in practice this approach of extracting substitutions has some shortcomings that we will discuss in this paper along with more robust mechanisms to obtain relevant instances. Let us demonstrate the Inst-Gen-Eq method by way of an example and point out the problems that we will address in the following sections.

Example 1. Consider the following unsatisfiable set of clauses.

$$f(x, y) \simeq f(y, x) \quad (1) \qquad f(a, b) \simeq g(c) \quad (3)$$
$$f(u, v) \not\simeq g(z) \ \vee \ u \simeq z \quad (2) \qquad a \not\simeq b \quad (4)$$

The ground abstractions of clauses (1) and (2) are $f(\bot, \bot) \simeq f(\bot, \bot)$ and $f(\bot, \bot) \not\simeq g(\bot) \ \vee \ \bot \simeq \bot$, respectively. Clauses (3) and (4) are ground and therefore identical to their ground abstractions. The ground abstractions of the first literals in each clause are satisfiable and can therefore be selected. With the following unit superposition proof we find the selected literals in clauses (2) and (3) to be inconsistent in first order.

$$\dfrac{\dfrac{\overset{(3)}{f(a,b) \simeq g(c)} \qquad \overset{(2)}{f(u,v) \not\simeq g(z)}}{g(c) \not\simeq g(z)}\,[a/u, b/v]}{\square}\,[c/z] \tag{$*$}$$

In order to make the inconsistency visible in the ground abstraction, we add an instance of clause (2) with the substitution $[a/u, b/v, c/z]$ obtained by composing the two substitutions in the proof. No substitution is applied to (3) because it is already ground.

$$f(a,b) \not\simeq g(c) \;\vee\; a \simeq c \tag{5}$$

We can prove another inconsistency in the set of selected literals.

$$\dfrac{\dfrac{\overset{(3)}{f(a,b) \simeq g(c)} \qquad \dfrac{\overset{(1)}{f(x,y) \simeq f(y,x)} \qquad \overset{(2)}{f(u,v) \not\simeq g(z)}}{f(v,u) \not\simeq g(z)}\,[u/x, v/y]}{g(c) \not\simeq g(z)}\,[a/v, b/u]}{\square}\,[c/z] \tag{$\dagger$}$$

After instantiating clauses (1) and (2) at the leaves of the proof with respective substitutions of $[b/x, a/y]$ and $[b/u, a/v, c/z]$ the ground abstraction consisting of clauses (3)-(7) becomes unsatisfiable. Again, the ground clause (3) cannot be instantiated.

$$f(b,a) \simeq f(a,b) \tag{6}$$

$$f(b,a) \not\simeq g(c) \;\vee\; b \simeq c \tag{7}$$

The main challenge we face in a practical implementation of the unit superposition calculus is the treatment of literal variants. Obviously, it is desirable to identify all literal variants in order to make the calculus less prolific and to avoid trivial non-termination. If all literal variants were treated separately, a commutativity axiom like $f(x,y) \simeq f(y,x)$ could infer an infinite number of variants of the literal $f(u,v) \not\simeq g(z)$ since we consider all literals to be variable disjoint.

On the other hand, the literal $f(u,v) \not\simeq g(z)$ occurs twice in the same branch of the second proof tree ($\dagger$) in Example 1 above. Linking both occurrences of the literal in the proof would collapse the proof tree into a graph with a cycle which is highly inconvenient in an implementation of the calculus. When composing substitutions on a branch of the proof tree, we would need to compose the substitution with itself an unbounded number of times. While in Example 1 the composition of the substitution $[u/x, v/y]$ with itself can only generate a finite number of instances, this is not the case for a substitution like $[f(x)/x]$.

To overcome these problems, we introduce labels for literals in unit superposition where we accumulate composed substitutions from inferences. The relevant instances can then be read off the label of the contradiction, thus obsoleting the need to trace a proof tree. Further, the labels will allow us to treat literal variants initially as disjoint while merging of literals is done in an explicit step that keeps track of literal variants merged in the label. This allows to uniformly treat literal variants in an implementation of the calculus and to include them in the usual heuristics in a given clause algorithm.

3 Set Labelled Unit Superposition

To each literal we attach a label and we consider literals with different labels to be distinct and also distinguish between variants of a literal. However, our calculus provides an explicit inference step to merge two variants of a literal into one with a label that joins both their labels. The labels of conclusions accumulate substitutions from their inferences, in this way eagerly extracting substitutions in a proof. The merging step combines the two proofs of a literal and its variant. Further inference steps are then simultaneously applied to both proofs.

The basic element of any literal label is a *closure* which is a pair of a clause C and a substitution θ, written as $C \cdot \theta$. In the first and simplest structure for labelling literals, a *label* $\mathcal{L}$ is a set of closures $\{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\}$. Given a substitution σ, the σ*-instance* of the label $\mathcal{L}$ is the label $\mathcal{L}\sigma = \{C_1 \cdot \theta_1\sigma, \ldots, C_n \cdot \theta_n\sigma\}$.

Initially, we create for each clause C, where L is the selected literal in C, a labelled literal $\{C \cdot []\} \colon L$ from the clause C and the empty substitution $[]$. We therefore distinguish between literals from different clauses in the beginning.

We modify the inference rules of the unit superposition calculus from Definition 1 to work on set labelled literals and add a merging rule for literals which are variants of each other. Note that the labels are only a mechanism for bookkeeping, they do not occur in the side conditions of the inferences.

Definition 2 (Set Labelled Unit Superposition).

Merging

$$\frac{\mathcal{L} \colon l \simeq r \qquad \mathcal{L}' \colon l' \simeq r'}{\mathcal{L} \cup \mathcal{L}'\sigma \colon l \simeq r}\,(\sigma) \qquad\qquad \frac{\mathcal{L} \colon l \not\simeq r \qquad \mathcal{L}' \colon l' \not\simeq r'}{\mathcal{L} \cup \mathcal{L}'\sigma \colon l \not\simeq r}\,(\sigma)$$

where σ is a renaming such that $l'\sigma = l$ and $r'\sigma = r$. The conclusion replaces the two premises.

Superposition

$$\frac{\mathcal{L} \colon l \simeq r \qquad \mathcal{L}' \colon s[l'] \simeq t}{\mathcal{L}\sigma \cup \mathcal{L}'\sigma \colon (s[r] \simeq t)\sigma}\,(\sigma) \qquad\qquad \frac{\mathcal{L} \colon l \simeq r \qquad \mathcal{L}' \colon s[l'] \not\simeq t}{\mathcal{L}\sigma \cup \mathcal{L}'\sigma \colon (s[r] \not\simeq t)\sigma}\,(\sigma)$$

where (i) $\sigma = \mathrm{mgu}(l, l')$, (ii) l' is not a variable, (iii) $l\sigma\rho \succ r\sigma\rho$ and (iv) $s[l']\sigma\rho \succ t\rho$ for some grounding substitution ρ.

Equality Resolution

$$\frac{\mathcal{L} \colon (l \not\simeq r)}{\mathcal{L}\sigma \colon \square}\,(\sigma)$$

where $\sigma = \mathrm{mgu}(l, r)$.

Having derived a labelled contradiction, we now do not need to trace a proof tree to obtain the relevant instances. Instead, we can read the clauses to be instantiated and

their respective relevant substitutions off the label: the set of *relevant instances* of a set labelled literal $\{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\} \colon L$ is the set $\{C_1\theta_1, \ldots, C_n\theta_n\}$.

Replacing extraction of substitutions from proofs with the labelling approach above preserves completeness of the instantiation procedure if *Inst-fairness* as in Lemma 6 in [4] is upheld. This requires unit superposition (i) to derive the contradiction from certain non-redundant selected literals and (ii) to generate instances such that the conflict on selected literal becomes redundant. We will show that our labelled calculus satisfies these two properties by a simulation argument.[1]

Unlabelled and labelled unit superposition in Definition 1 and Definition 2, respectively, have the same side conditions and the same premises lead to the same conclusion. We can therefore state the following lemma from which (i) follows.

Lemma 1. *Any unlabelled unit superposition derivation can be stepwise simulated by set labelled unit superposition using only superposition and equality resolution.*

The required instances in (ii) are provided by the relevant instances extracted from a proof of the contradiction. We show that the relevant instances of an unlabelled proof of a literal are contained in the relevant instances of a label of the literal in every corresponding labelled proof. We first consider labelled unit superposition without merging and in a second step show that inserting merging inferences is compatible.

Lemma 2. *Let the literal L be derived by unlabelled unit superposition and let the relevant instances extracted from the proof of L be $C_1\theta_1, \ldots, C_n\theta_n$. The stepwise simulation by set labelled unit superposition yields the labelled literal $\mathcal{L} \colon L$ where $\mathcal{L}$ contains the set of closures $\{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\}$.*

To finish the argument we have to include merging inferences which can occur at any step of a labelled unit superposition derivation. The conclusion of a merging inference replaces its premises and renames the literals to make them identical. We therefore must ensure that after merging literals the relevant instances that were in the label of a premise remain in the label of the conclusion.

Lemma 3. *Let L be a literal and let the relevant instances extracted from its proof be $C_1\theta_1, \ldots, C_n\theta_n$. After a merging inference with the labelled literal $\mathcal{L} \colon L$ as the left premise (the right premise, respectively) the label of the conclusion contains the closures $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$ (respectively $C_1 \cdot \theta_1\sigma, \ldots, C_n \cdot \theta_n\sigma$).*

Finally if unit superposition is applied in a fair way, that is every non-redundant inference is drawn eventually, we can state the completeness theorem which is based on results from [4].

Corollary 1. *A fair labelled unit superposition process with instantiation of relevant instances from contradictions yields an Inst-fair saturation process.*

Let us resume our running example, demonstrating the merging inference rule before we point out some disadvantages of set labels and move on to different label structures in the next sections.

[1] For proofs see `http://www.cs.man.ac.uk/~sticksec/LPAR2010-Full.pdf`

Example 2. We draw an inference which corresponds to the first inference in proof (†) in Example 1 from the selected literals in clauses (1) and (2).

$$\dfrac{\overset{(1)}{\{(1)\cdot[]\}\colon f(x,y)\simeq f(y,x)} \qquad \overset{(2)}{\{(2)\cdot[]\}\colon f(u,v)\not\simeq g(z)}}{\{(1)\cdot[u/x,v/y],(2)\cdot[]\}\colon f(v,u)\not\simeq g(z)}\,[u/x,v/y] \tag{†′}$$

We note that applying the substitution $[u/x, v/y]$ to the closure $(2)\cdot[]$ results in the same closure as the variables x and y in the domain of the substitution do not occur in the closure. The conclusion $f(v,u)\not\simeq g(z)$ is a variant of the right premise and we can merge the two literal variants into one which replaces the distinct variants.

$$\dfrac{\overset{(2)}{\{(2)\cdot[]\}\colon f(u,v)\not\simeq g(z)} \qquad \overset{(†′)}{\{(1)\cdot[u/x,v/y],(2)\cdot[]\}\colon f(v,u)\not\simeq g(z)}}{\{(2)\cdot[],(1)\cdot[v/x,u/y],(2)\cdot[v/u,u/v]\}\colon f(u,v)\not\simeq g(z)}\,[v/u,u/v] \tag{‡}$$

With superposition and equality resolution which are the last steps in both proofs $(*)$ and (†) in Example 1, we derive the contradiction.

$$\dfrac{\dfrac{\overset{(3)}{\{(3)\cdot[]\}\colon f(a,b)\simeq g(c)} \qquad \overset{(‡)}{\{(2)\cdot[],(1)\cdot[v/x,u/y],(2)\cdot[v/u,u/v]\}\colon f(u,v)\not\simeq g(z)}}{\{(3)\cdot[],(2)\cdot[a/u,b/v],(1)\cdot[b/x,a/y],(2)\cdot[b/u,a/v]\}\colon g(c)\not\simeq g(z)}\,[a/u,b/v]}{\{(3)\cdot[],(2)\cdot[a/u,b/v,c/z],(1)\cdot[b/x,a/y],(2)\cdot[b/u,a/v,c/z]\}\colon\square}\,[c/z] \tag{∗′}$$

The instances of the clauses in the label of the contradiction with their respective substitutions are exactly the instances (5)-(7) in Example 1 and unsatisfiability can be shown on the ground abstraction of clauses (1)-(7).

We can exploit the observation that set labels can become saturated to prove that set labelled unit superposition is a decision procedure for the Bernays-Schönfinkel fragment of first-order clause logic with equality. We first need to introduce some terminology motivated by the usual idea in theorem proving that clauses are equal modulo renaming. We extend this notion to closures and labelled literals in the following ways.

Two closures $C\cdot\theta$ and $C'\cdot\theta'$ are *equivalent up to renaming away* from a set of variables $\mathcal{V}$ if there exists a renaming μ where $\mathrm{rng}(\mu)\cap\mathcal{V}=\emptyset$ such that $C\cdot\theta = C'\mu\cdot\mu^{-1}\theta'\mu$. Intuitively we want the closures to be equal if C and C' as well as $C\theta$ and $C'\theta'$ are equal up to renaming, therefore we have to "pull out" the renaming μ from the substitution θ'. Further, we want to make the clauses variable-disjoint from $\mathcal{V}$, which becomes obvious in the next step.

For a labelled literal $\mathcal{L}\colon L$ we do not distinguish closures in $\mathcal{L}$ equivalent up to renaming away from $\mathrm{var}(L)$. Clauses in the label $\mathcal{L}$ are made variable-disjoint from L and without loss of generality we assume that for a closure $C\cdot\theta$ it is $\mathrm{dom}(\theta)\subseteq\mathrm{var}(C)$.

We say that a set of closures $\mathcal{S}$ is *equivalent up to renaming away* from a set of variables $\mathcal{V}$ to a set of closures $\mathcal{S}'$ if for every closure in $\mathcal{S}$ there is a closure equivalent up to renaming away from $\mathcal{V}$ in $\mathcal{S}'$ and vice versa.

Finally, two labelled literals $\mathcal{L}\colon L$ and $\mathcal{L}'\colon L'$ are *equivalent up to renaming* if there is a renaming ρ such that $L = L'\rho$ and $\mathcal{L}$ is equivalent to $\mathcal{L}'\rho$ up to renaming away from $\mathrm{var}(L)$. We then do not distinguish between labelled literals equivalent up to renaming.

Theorem 1. *Inst-Gen-Eq with set labelled unit superposition is a decision procedure for the Bernays-Schönfinkel fragment of first-order logic with equality.*

Proof. There is only a finite number of labelled literals that are not equivalent up to renaming and thus only a finite number of relevant instances from labels of contradictions. The set labelled unit superposition calculus can therefore derive only a finite number of distinct set labelled literals.

As discussed, set labels make available the relevant instances directly, thus obsoleting the need to trace a proof tree. The proof of a literal cannot be reconstructed from its label as the union of labels in merging and superposition inferences loses the proof structure. However, the relevant instances are all that is needed to witness the inconsistency of the selection in the ground abstraction. Further, the merging inference can combine several proofs with their common parts factored out in the union of the labels.

4 Redundancy Elimination and Selection Changes

Although set labels are a concise and powerful enough mechanism in many practical cases, they show a weakness when we consider redundancy that occurs in the incremental process of instantiation. Set labels collect clauses at the leaves of proofs and accumulate respective relevant substitutions. Merging inferences combine superposition proofs and the conclusion contains closures from several proofs. When a leaf clause becomes redundant with the accumulated relevant substitution, every proof with the clause at a leaf is redundant and all leaf clauses in this proof can be eliminated. However, in a set label we cannot separate out all closures corresponding to the redundant proof from a set label since the structure is lost when two proofs are merged.

Example 3. Let us assume that the unlabelled unit superposition proof (†) in Example 1 becomes redundant due to the accumulated substitution applied to (2).[2] Let us further assume proof $(*)$ is not redundant. In the labelled unit superposition calculus in Example 2, both proofs were merged and we had $\{(3) \cdot [], (2) \cdot [a/u, b/v, c/z], (1) \cdot [b/x, a/y], (2) \cdot [b/u, a/v, c/z]\}$ as the label of the contradiction.

We would obtain the set label $\{(2) \cdot [b/u, a/v, c/z], (1) \cdot [b/x, a/y], (3) \cdot []\}$ from the redundant proof (†) and we want to eliminate these redundant closures from the set label of the merged proofs. However, the set label from the non-redundant proof $(*)$ is $\{(3) \cdot [], (2) \cdot [a/u, b/v, c/z]\}$ and we have to retain these closures.

As the information about the proof structure cannot be recovered from a set label, we cannot eliminate all closures from the a set label. In particular, we would need to know that the proofs overlap on closure $(3) \cdot []$ and not on $(1) \cdot [b/x, a/y]$.

A similar problem arises from changes in the selection function. The model of the ground abstraction may change in a way that a different literal has to be selected in a clause than before. In that case, all proofs with the previously selected literal at a leaf should be eliminated. Again, we want to remove a subset of the clauses in the label of a

[2] See `http://www.cs.man.ac.uk/~sticksec/LPAR2010-Full.pdf` for an extended example with concrete redundancy

literal and it is not possible to determine if a clause has to be kept in the label as it may well be from the label of a non-redundant proof that was merged.

The cause of the problem is that we use the set union for combining labels in both the merging inference and in the superposition inference. In the next section we will present a different label structure that preserves the shape of proofs by using two different operations in merging and superposition.

Let us finally note that set labels are still a useful sound and complete mechanism. Unit superposition with set labels merely generates more instances of clauses than strictly necessary while adding these instances does not harm soundness nor completeness. An instance of a clause is a sound consequence of the clause. Although we cannot determine the full subset to be eliminated from a label, we can safely remove each clause from a label which has become redundant with its substitution or where the selection has changed. In Section 7 we will evaluate an implementation of set labels with this restricted elimination against the more powerful elimination in the next section.

5 Tree labelled Unit Superposition

In order to eliminate redundancy in a labelled unit superposition calculus as described above, we need to preserve a certain Boolean structure in the label. To this end we can regard a closure $C \cdot \theta$ as a propositional variable. A merging inference corresponds to a disjunction and a superposition to a conjunction of labels. Eliminating parts of a label then means assigning false to propositional variables in the tree where the corresponding closures have become redundant and simplifying the Boolean structure.

Let us write the disjunction of two labels as $\mathcal{T}_1 \sqcup \mathcal{T}_2$ and the conjunction of two labels as $\mathcal{T}_1 \sqcap \mathcal{T}_2$. A tree label is then either a closure $C \cdot \theta$, a conjunction $\bigsqcap_{i=1}^{n} \mathcal{T}_i$ or a disjunction $\bigsqcup_{i=1}^{n} \mathcal{T}_i$ of n tree labels $\mathcal{T}_1, \ldots, \mathcal{T}_n$. This structure is isomorphic to an AND/OR tree where all non-leaf nodes are either labelled as AND nodes or OR nodes. AND and OR nodes alternate on each level of the tree such that AND nodes only have OR nodes as successors and vice versa for OR nodes. We call this label a tree label.

Definition 3. *A* tree label *is an AND/OR tree where each leaf is a closure $C \cdot \theta$. The* σ-instance *of a tree label $\mathcal{T}$ is the AND/OR tree $\mathcal{T}$ with the substitution σ applied at each leaf such that $C \cdot \theta$ becomes $C \cdot \theta\sigma$.*

In order to eliminate from a tree label a closure that has become redundant, we assign false to the propositional variable and simplify the tree with Boolean operations.

Definition 4. *The $C \cdot \theta$-*restriction *$\mathcal{T}|_{C\cdot\theta}$ of an AND/OR tree $\mathcal{T}$ is the tree obtained by replacing every occurrence of $C \cdot \theta$ in $\mathcal{T}$ with the constant* false *and recursively simplifying the tree using the rules (i) $\mathsf{false} \sqcup \mathcal{U} \to \mathcal{U}$ and (ii) $\mathsf{false} \sqcap \mathcal{U} \to \mathsf{false}$.*

The strength of tree labels when compared to set labels is the precise elimination of redundancy by restriction. If the closure $C \cdot \theta$ has become redundant, then we can simplify the tree label $C \cdot \theta \sqcup \mathcal{T}$ to $\mathcal{T}$ and the label $C \cdot \theta \sqcap \mathcal{T}$ to the empty label. Literals with the latter label are redundant and can be discarded.

We now define a unit superposition calculus with different operators to combine labels in the merging and the superposition inference, namely $\sqcup$ and $\sqcap$, respectively.

Definition 5 (Tree Labelled Unit Superposition).

Restriction

$$\frac{\mathcal{T} : L}{\mathcal{T}|_{C\cdot\theta} : L}$$

where $C \cdot \theta$ is redundant. The label of the literal is replaced with its restriction.

Merging

$$\frac{\mathcal{T} : l \simeq r \qquad \mathcal{T}' : l' \simeq r'}{\mathcal{T} \sqcup \mathcal{T}'\sigma : l \simeq r}\,(\sigma) \qquad \frac{\mathcal{T} : l \not\simeq r \qquad \mathcal{T}' : l' \not\simeq r'}{\mathcal{T} \sqcup \mathcal{T}'\sigma : l \not\simeq r}\,(\sigma)$$

where σ is a renaming such that $l'\sigma = l$ and $r'\sigma = r$. The conclusion replaces the two premises.

Superposition

$$\frac{\mathcal{T} : l \simeq r \qquad \mathcal{T}' : s[l'] \simeq t}{(\mathcal{T} \sqcap \mathcal{T}')\sigma : (s[r] \simeq t)\sigma}\,(\sigma) \qquad \frac{\mathcal{T} : l \simeq r \qquad \mathcal{T}' : s[l'] \not\simeq t}{(\mathcal{T} \sqcap \mathcal{T}')\sigma : (s[r] \not\simeq t)\sigma}\,(\sigma)$$

where (i) $\sigma = \mathrm{mgu}(l, l')$, (ii) l' is not a variable, (iii) $l\sigma\rho \succ r\sigma\rho$ and (iv) $s[l']\sigma\rho \succ t\rho$ for some grounding substitution ρ.

Equality Resolution

$$\frac{\mathcal{T} : (l \not\simeq r)}{\mathcal{T}\sigma : \square}\,(\sigma)$$

where $\sigma = \mathrm{mgu}(l, r)$.

As in the set labelled unit superposition calculus, we start with the selected literals which are labelled with the respective clauses they are selected in. Upon finding the contradiction we generate the instances of all clauses at the leaves of the tree. The set of *relevant instances* of a tree label is the set of closures occurring at leaves of the AND/OR tree.

In order to show that tree labelled unit superposition can replace set labelled unit superposition and in turn unlabelled unit superposition with extraction of relevant instances from proofs, we need to extend the simulation argument from Section 3.

Tree labelled unit superposition can stepwise simulate set labelled unit superposition using only superposition, merging and equality resolution inferences and the relevant instances in set labels are identical to the relevant instances in the tree label. Therefore Lemmas 1, 2 and 3 apply for tree labels as well, but we additionally have to deal with the restriction inference and show that it preserves completeness using the following lemma.

Lemma 4. *A restriction inference on $C \cdot \theta$ eliminates exactly those closures from the tree label which occur in corresponding unlabelled proofs, redundant due to $C \cdot \theta$.*

The lemma follows by induction over the proof structure the tree labelled literal was derived from. Let us give an example to illustrate elimination by restriction and the subsequent simplification of the AND/OR tree.

Example 4. The contradiction in the set labelled unit superposition proof ($*'$) from Example 2 has the set label

$$\Big\{(3)\cdot[],(2)\cdot[a/u,b/v,c/z],(1)\cdot[b/x,a/y],(2)\cdot[b/u,a/v,c/z]\Big\}.$$

In a tree labelled unit superposition proof, we obtain the label

$$(3)\cdot[]\sqcap\Big((2)\cdot[a/u,b/v,c/z]\sqcup\Big((1)\cdot[b/x,a/y]\sqcap(2)\cdot[b/u,a/v,c/z]\Big)\Big)$$

which preserves the structure of the two proofs that were merged.

If we were to eliminate $(2)\cdot[b/u,a/v,c/z]$ which corresponds to the leaf literal $f(u,v)\not\simeq g(z)$ in proof (†) in Example 1, the tree label becomes

$$(3)\cdot[]\sqcap\Big((2)\cdot[a/u,b/v,c/z]\sqcup\Big((1)\cdot[b/x,a/y]\sqcap\mathsf{false}\Big)\Big)=$$
$$(3)\cdot[]\sqcap(2)\cdot[a/u,b/v,c/z].$$

Eliminating $(2)\cdot[a/u,b/v,c/z]$ which corresponds to $f(u,v)\not\simeq g(z)$ in proof ($*$) in Example 1 leaves us with

$$(3)\cdot[]\sqcap\Big(\mathsf{false}\sqcup\Big((1)\cdot[b/x,a/y]\sqcap(2)\cdot[b/u,a/v,c/z]\Big)\Big)=$$
$$(3)\cdot[]\sqcap(1)\cdot[b/x,a/y]\sqcap(2)\cdot[b/u,a/v,c/z].$$

Both tree labels then lead to exactly the instances that were generated from the separate proofs in Example 1.

6 OBDD Labelled Unit Superposition

Two labelled literals $\mathcal{L}_1\colon L$ and $\mathcal{L}_2\colon L$ are identical if their labels are. Moreover, since labels encode proofs and certain proofs are isomorphic, we can generalise the notion of identity on labelled literals to equivalence based on logical equivalence of Boolean formulae. Exploiting equivalence of labels in a labelled unit superposition procedure is not only an important simplification step, in some cases it is essential for termination as we will show on our running example.

Equivalence of two set labels can easily be checked: if their sets are equal, their relevant instances are equal and the literals do not need to be distinguished. Here, the property that sets are unordered collections of elements leads to a natural normal form where equivalence of labels can be checked efficiently. However, the situation is not so easy for tree labels since they are not produced in a normal form. The sequence of merging and superposition inferences determines the shape of the tree which makes comparing tree labels by their shape unusable except in simple cases.

Standard normal forms of Boolean formulae are the disjunctive and conjunctive normal forms (DNF and CNF) which are unfortunately frequently exponential in the

size of the original formula. However, approaches like the definitional transformation, which introduces new variables for subterms of the original formula, do not produce a unique normal form, which makes checking equivalence of labels more difficult.

In this section we propose tree labels based on ordered binary decision diagrams (OBDDs) which offer particularly promising features, above all unique normal forms and checking of equivalence in constant time. An OBDD is a graph with common subtrees shared, it provides a compact and well-understood normal form of Boolean formulae. OBDDs are used in similar contexts to encode Boolean structures, e.g. [9].

Definition 6. *An* OBDD label $\mathcal{B}$ *is an OBDD where each node is a closure* $C \cdot \theta$*. The* σ-instance *of an OBDD label* $\mathcal{B}$ *is the OBDD* $\mathcal{B}\sigma$ *where each closure* $C \cdot \theta$ *is replaced with* $C \cdot \theta\sigma$*.*

We remark that σ-instantiation of an OBDD may require changing the variable ordering in the OBDD and thus a reordering of the OBDD.

Definition 7. *The* $C \cdot \theta$-restriction $\mathcal{B}|_{C \cdot \theta}$ *of an OBDD label is obtained by replacing each node* $C \cdot \theta$ *in* $\mathcal{B}$ *with* false *and reducing the OBDD.*

The inference rules of OBDD labelled unit superposition are in straightforward analogy to tree labelled unit superposition in Definition 5 where tree labels are replaced with OBDDs. We obtain relevant instances from an OBDD label in the obvious way: the set of *relevant instances* of an OBDD label $\mathcal{B}$ is the set of all nodes $C \cdot \theta$ in $\mathcal{B}$.

Let us further discuss our running example where keeping the Boolean structure of a tree label in an OBDD normal form prevents non-termination in the case of not normalised tree labels.

Example 5. We notice that the equation $f(x, y) \simeq f(y, x)$ is not orientable in any simplification ordering and must therefore be applied in both directions. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the labels of $f(x, y) \simeq f(y, x)$ and $f(u, v) \not\simeq g(z)$, respectively.

We draw a first superposition inference between the literals with the equation in the given orientation

$$\frac{\begin{array}{cc}(1) & (2)\\ \mathcal{T}_1 \colon f(x, y) \simeq f(y, x) & \mathcal{T}_2 \colon f(u, v) \not\simeq g(z)\end{array}}{\mathcal{T}_1[u/x, v/y] \sqcap \mathcal{T}_2[] \colon f(v, u) \not\simeq g(z)}\,[u/x, v/y] \qquad (i)$$

and merge the conclusion with the premise

$$\frac{\begin{array}{cc}(2) & (i)\\ \mathcal{T}_2[] \colon f(u, v) \not\simeq g(z) & \mathcal{T}_1[u/x, v/y] \sqcap \mathcal{T}_2[] \colon f(v, u) \not\simeq g(z)\end{array}}{\mathcal{T}_2[] \sqcup \Big(\mathcal{T}_1[v/x, u/y] \sqcap \mathcal{T}_2[v/u, u/v]\Big) \colon f(u, v) \not\simeq g(z)}\,[v/u, u/v] \qquad (ii)$$

A second superposition with the equation reversed

$$\frac{\begin{array}{cc}(1) & (2)\\ \mathcal{T}_1 \colon f(y, x) \simeq f(x, y) & \mathcal{T}_2 \colon f(u, v) \not\simeq g(z)\end{array}}{\mathcal{T}_1[v/x, u/y] \sqcap \mathcal{T}_2[] \colon f(v, u) \not\simeq g(z)}\,[v/x, u/y] \qquad (iii)$$

results in the same conclusion with a different label. We merge it again

$$\frac{\overset{(2)}{\mathcal{T}_2[]\colon f(u,v) \not\approx g(z)} \qquad \overset{(iii)}{\mathcal{T}_1[v/x,u/y] \sqcap \mathcal{T}_2[]\colon f(v,u) \not\approx g(z)}}{\mathcal{T}_2[] \sqcup \Big(\mathcal{T}_1[u/x,v/y] \sqcap \mathcal{T}_2[v/u,u/v]\Big)\colon f(u,v) \not\approx g(z)}\,[v/u,u/v] \qquad (iv)$$

and obtain the following tree label from (ii) and (iv)

$$\mathcal{T}_2[] \sqcup \Big(\mathcal{T}_1[v/x,u/y] \sqcap \mathcal{T}_2[v/u,u/v]\Big) \sqcup \Big(\mathcal{T}_1[u/x,v/y] \sqcap \mathcal{T}_2[v/u,u/v]\Big).$$

Let us abbreviate this label to

$$\mathcal{T}_2^1 \sqcup \Big(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^{-1}\Big) \sqcup \Big(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^{-1}\Big) \qquad \text{(a)}$$

using the superscript ^{1} to denote the substitutions $[]$ and $[u/x,v/y]$ as well as $^{-1}$ for $[v/x,u/y]$ and $[v/u,u/v]$. The corresponding set label is

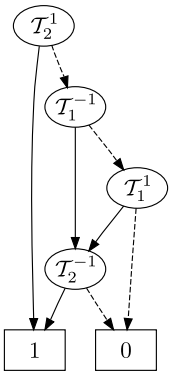
$$\Big\{\mathcal{T}_2^1, \mathcal{T}_2^{-1}, \mathcal{T}_1^1, \mathcal{T}_1^{-1}\Big\}.$$

It is now necessary to repeat inferences (i) and (iii) for $f(v,u) \not\approx g(z)$ with the label $\mathcal{T}_2$ being (a). We note that for the substitutions applied to $\mathcal{T}_2$ in the merging, we have $\mathcal{T}_1^{-1}[v/u,u/v] = \mathcal{T}_1^1$, $\mathcal{T}_1^1[v/u,u/v] = \mathcal{T}_1^{-1}$, $\mathcal{T}_2^{-1}[v/u,u/v] = \mathcal{T}_2^1$ and $\mathcal{T}_2^1[v/u,u/v] = \mathcal{T}_2^{-1}$.

The label of the conclusion has the structure of (a) where $\mathcal{T}_2^1$ is substituted by (a) and $\mathcal{T}_2^{-1}$ by (a) with the substitution $[v/u,u/v]$ applied.

$$\begin{aligned}\mathcal{T}_2^1 \sqcup \Big(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^{-1}\Big) \sqcup \Big(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^{-1}\Big) \sqcup \\ \Big(\mathcal{T}_1^{-1} \sqcap \Big(\mathcal{T}_2^{-1} \sqcup \Big(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^1\Big) \sqcup \Big(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^1\Big)\Big)\Big) \sqcup \\ \Big(\mathcal{T}_1^1 \sqcap \Big(\mathcal{T}_2^{-1} \sqcup \Big(\mathcal{T}_1^1 \sqcap \mathcal{T}_2^1\Big) \sqcup \Big(\mathcal{T}_1^{-1} \sqcap \mathcal{T}_2^1\Big)\Big)\Big)\end{aligned} \qquad \text{(b)}$$

The set label does not change as no new leaves are added in the tree label. Literals with identical labels are identical and therefore no further inferences are necessary.
As the tree labels (a) and (b) have a different structure from the way they were built up during the inferences, they are distinct and we would continue with inferences, obtaining ever-growing labels in the conclusion.
If we, however, transform both labels to an OBDD using the same ordering, we obtain the relatively simple OBDD shown to the right. Just as for set labels we do not need to generate further inferences from here.



OBDD labels are in a normal form as set labels are, therefore we can state a variant of Theorem 1

Problems	solved	not solved	fastest
set	2006	259	601
tree	1983	282	699
OBDD	1512	753	93

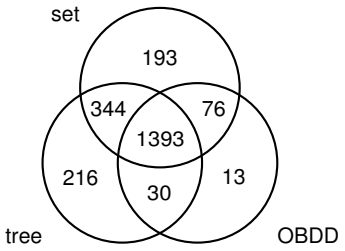


Fig. 1. Comparing labels on the number of solved equational problems out of 9054 in total: set labels solved 2006 problems and were fastest on 601 problems, while 259 problems were only solved with other labels. 193 problems could be solved with set labels and not with tree or OBDD labels, 30 problems were not solved with set labels but both with tree and OBDD labels and 1393 problems were solved with all three labels.

Theorem 2. *Inst-Gen-Eq with OBDD labelled unit superposition is a decision procedure for the Bernays-Schönfinkel fragment of first-order logic with equality.*

Proof. There is only a finite number of distinct closures and therefore only a finite number of OBDDs built from these closures. Therefore there is only a finite number of distinct OBDD labelled literals and the OBDD labelled unit superposition calculus will therefore terminate after a finite number of inference steps.

7 Implementation and Evaluation

We have implemented set, tree and OBDD labels in our iProver-Eq system (see [7]) and evaluated it with the TPTP benchmark library v4.0.1. We have used a cluster of Intel Xeon Quad Core machines with 2.33GHz and 2GB of memory limit and ran each of the 13783 problems for at most 120 seconds. In total, 4848 problems were solved by at least one label implementation and 3970 problems were solved by all three implementations.

As expected, the performance on non-equational problems was equal in all three label implementations, therefore we only focus on the 9054 problems with at least one equation, see Figure 1.

The results show that set and tree labels exhibit a comparable performance on both the overall number of solved problems and the number of problems that were solved fastest in the implementation. The number of problems solved only with set and tree labels (193 and 216) are significant, as well as the number of problems solved with other labels but not with the respective implementation (259 and 282). In a direct comparison, there are 195 problems where tree labels are more than twice as fast as set labels, whereas vice versa set labels are twice as fast on only 70 problems.

Despite the fact that OBDD labels provide a normal form, efficient checking for label equivalence and precise elimination of redundancy, in practice they remain considerably weaker than trees and sets. Their performance is mainly hit by the effort spent building OBDD labels which can become rather large. In the problems that were solved, OBDDs were well-behaved so that the number of nodes in OBDDs was in most cases much less than quadratic in the number of variables, i.e. closures in labels. Problems

that were not solved in the time limit mostly had either a large number of closures (up to 50,000) or the Boolean structure had to be represented with a large number of nodes (many with several millions). Nevertheless, there are 119 problems solved with OBDD labels which are not solved with both of the other label implementations. Further, on 93 problems OBDD labels are faster than both the other labels, which include 9 problems with a runtime greater than one second where OBDD labels are significantly faster.

The results make hybrid approaches, such as a combination of tree and set labels, look promising. We will also further investigate how to improve OBDD labels with techniques exploiting the specific structure of labels.

8 Conclusion

In this paper we introduced a labelled superposition calculus for efficient instance generation for equational reasoning in the Inst-Gen framework. We investigated and evaluated several label structures based on sets, AND/OR trees and OBDDs. Our implementation and experimental results show that our labelled approach has a promising potential. We observe that different label structures are complementary in performance on many problems which indicates further investigation is needed regarding possible combinations of these techniques. In further work we will also explore possibilities to make use of the information in labels in other calculi and applications, for example for query answering.

References

1. Baumgartner, P.: Logical Engineering with Instance-Based Methods. In: CADE-21. LNAI, vol. 4603, pp. 404–409. Springer (2007)
2. Baumgartner, P., Tinelli, C.: The Model Evolution Calculus with Equality. In: CADE-20. LNCS, vol. 3632, pp. 392–408. Springer (2005)
3. Eiter, T., Faber, W., Traxler, P.: Testing Strong Equivalence of Datalog Programs - Implementation and Examples. In: LPNMR 2005. LNCS, vol. 3662, pp. 437–441. Springer (2005)
4. Ganzinger, H., Korovin, K.: Integrating Equational Reasoning into Instantiation-Based Theorem Proving. In: CSL 2004. LNCS, vol. 3210, pp. 71–84. Springer (2004)
5. Hustadt, U., Motik, B., Sattler, U.: Reducing SHIQ- Description Logic to Disjunctive Datalog Programs. In: KR 2004. pp. 152–162. AAAI Press (2004)
6. Korovin, K.: Instantiation-Based Automated Reasoning: From Theory to Practice. In: CADE-22. LNCS, vol. 5663, pp. 163–166. Springer (2009)
7. Korovin, K., Sticksel, C.: iProver-eq – An Instantiation-based Theorem Prover with Equality. In: IJCAR 2010. LNCS, vol. 6173, pp. 196–202. Springer (2010)
8. Letz, R., Stenz, G.: Integration of Equality Reasoning into the Disconnection Calculus. In: TABLEAUX 2002. pp. 176–190 (2002)
9. de Moura, L., Bjørner, N.: Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. In: IJCAR 2008. LNCS, vol. 5195, pp. 410–425. Springer (2008)
10. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning. Elsevier (1999)
11. Pérez, J.A.N., Voronkov, A.: Encodings of Bounded LTL Model Checking in Effectively Propositional Logic. In: CADE-21. pp. 346–361 (2007)

A Proofs and Examples

Lemma 2. *Let the literal L be derived by unlabelled unit superposition and let the relevant instances extracted from the proof of L be $C_1\theta_1, \ldots, C_n\theta_n$. The stepwise simulation by set labelled unit superposition yields the labelled literal $\mathcal{L}\colon L$ where $\mathcal{L}$ contains the set of closures $\{C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n\}$.*

Proof. Stepwise simulation of unlabelled superposition only contains superposition and equality resolution inferences and we proceed by induction over the proof of L. Let L_1 and L_2 be the premises of a superposition inference to L with an mgu σ and let $\theta_1, \ldots, \theta_n$ and $\rho_1, \ldots, \rho_k$ be the relevant substitutions for clauses $C_1, \ldots, C_n$ and $D_1, \ldots, D_k$ in the proofs of L_1 and L_2, respectively. The relevant substitutions in the proof of L are $\theta_1\sigma, \ldots, \theta_n\sigma, \rho_1\sigma, \ldots, \rho_k\sigma$ for the clauses $C_1, \ldots, C_n, D_1, \ldots, D_k$. We can assume that the labelled literals $\mathcal{L}_1\colon L_1$ and $\mathcal{L}_2\colon L_2$ are the premises of a labelled superposition inference and that $\mathcal{L}_1$ and $\mathcal{L}_2$ contain the closures $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$ and $D_1 \cdot \rho_1, \ldots, D_k \cdot \rho_k$, respectively. Finally the label of the conclusion $(\mathcal{L}_1 \cup \mathcal{L}_2)\sigma\colon L$ contains the closures $C_1 \cdot \theta_1\sigma, \ldots, C_n \cdot \theta_n\sigma, D_1 \cdot \rho_1\sigma, \ldots, D_k \cdot \rho_k\sigma$ whose relevant instances are the relevant instances extracted from the unlabelled proof above. A similar argument holds for equality resolution.

Lemma 3. *Let L be the premise of a merging inference with the relevant instances $C_1\theta_1, \ldots, C_n\theta_n$ extracted from the proof of L. After a merging inference with the labelled literal $\mathcal{L}\colon L$ as the left premise (the right premise) the label of the conclusion contains the closures $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$ (respectively $C_1 \cdot \theta_1\sigma, \ldots, C_n \cdot \theta_n\sigma$)*

Proof. We again proceed by induction. From Lemma 2 we have that the labelled literal $\mathcal{L}\colon L$ from the stepwise simulation of the proof of L contains $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$. If $\mathcal{L}\colon L$ is the left premise of a merging inference, the conclusion $\mathcal{L} \cup \mathcal{L}'\theta\colon L$ contains the closures $C_1 \cdot \theta_1, \ldots, C_n \cdot \theta_n$. If $\mathcal{L}\colon L$ is the right premise of a merging inference, the conclusion $\mathcal{L}' \cup \mathcal{L}\theta\colon L$ contains the closures $C_1 \cdot \theta_1\sigma, \ldots, C_n \cdot \theta_n\sigma$.

Example 6 (extended example 3). Let us extend the running example by assuming the following instance of (2) has been generated

$$f(b, v) \not\simeq g(c) \ \vee \ b \simeq c \tag{2'}$$

and (2) thus has the substitution $[b/u, c/z]$ in its dismatching constraint.

Proof (†) from Example 1 is now redundant as the relevant substitution for (2), $[b/x, a/y, c/z]$, is blocked by its dismatching constraint. Proof $(*)$ is not redundant, there the relevant substitution for (2), $[a/x, b/y, c/z]$, is not more general than $[b/x, c/z]$.

In the labelled unit superposition calculus in Example 2, both proofs were merged and we had $\{(3) \cdot [], (2) \cdot [a/u, b/v, c/z], (1) \cdot [b/x, a/y], (2) \cdot [b/u, a/v, c/z]\}$ as the label of the contradiction.

We would obtain the set label $\{(2) \cdot [b/u, a/v, c/z], (1) \cdot [b/x, a/y], (3) \cdot []\}$ (as in Example 3) from the redundant proof (†) and we want to eliminate these redundant closures from the set label of the merged proofs. However, the set label from the non-redundant proof $(*)$ is $\{(3) \cdot [], (2) \cdot [a/u, b/v, c/z]\}$ and we have to retain these closures.