System Description: SPASS-FD

Matthias Horbach

University of New Mexico, Albuquerque, U.S.A.
horbach@cs.unm.edu

Abstract. Using a constrained superposition calculus and a disunification procedure, it is possible to employ superposition-based first-order reasoners for reasoning not only about all models of a first-order theory, but also about all models over a specific finite domain and often as well about the perfect models of the theory (or the unique minimal model in case of a Horn theory), both of which are second-order problems.
In this paper, I describe the tool SPASS-FD, an extension of the first-order prover SPASS that equips SPASS with disunification and with fixed domain and minimal model theorem proving capabilities.

1 Introduction

Saturation-based calculi such as superposition [10] can be instantiated to decision procedures for many decidable fragments of first-order logic. Superposition provers are state of the art in first-order theorem proving, i.e. in deciding whether a given set of clauses has a model. Often however, any model is not good enough: When describing real-world systems, the domain of the admissible models is usually predefined as the Herbrand universe over the symbols appearing in the description. Moreover, the system description will mostly state what the system can do, leaving what it cannot do implicit: The intended semantics of the description is a minimal model semantics. For formal reasoning about the system, the implicit information about the minimal model has to be recovered.

Originally, this line of thought was developed to handle negation in logic programming [2]. In automated theorem proving, it can be used for example to reason about the so-called contexts in the model evolution calculus [1], or about models described by sets of atoms with exceptions [5], or in general for any kind of inductive problems.

Aiming at the long-standing goal to harvest the power of first-order reasoning for this setting [4], Horbach and Weidenbach [9,7] developed a superposition-based algorithm that allows for fixed domain reasoning with a saturation-based first-order prover. Later, this was extended using predicate completion [2,4,6] to systems for minimal model reasoning that decide, e.g., validity of queries with one quantifier alternation for the above-mentioned contexts [8,7,6].

The tool SPASS-FD is an implementation on top of the automated theorem prover SPASS [14] of (i) the calculi for saturation-based reasoning with respect to fixed domains and minimal models as described in [7] (including the decision procedure for contexts), and (ii) the underlying disunification procedure from

[6]. SPASS provides a powerful superposition-based saturation machinery, which makes this prover well-suited as a basis for the implementation. SPASS-FD can be downloaded the SPASS homepage or from `cs.unm.edu/~horbach/software/`.

2 Theoretical Background

For the sake of simplicity, I will in the following always assume all clauses to be predicative and Horn.

Disunification and Predicate Completion. *Disunification* [3,6] is an extension of the idea of unification from equations to arbitrary equational formulas. Technically, it is a rewrite system for quantifier elimination in the empty theory.

The most obvious use of disunification is as a decision procedure for satisfiability of equational formulas. A second and for the current application even more important one is *predicate completion* [2], where a set N of clauses is extended to a set N' such that N' has only one Herbrand model, namely the minimal model of N. Predicate completion is one of the main tools to connect first-order, fixed domain and minimal model reasoning [4].

Fixed Domain and Minimal Model Reasoning. Superposition is an established decision procedure for a variety of first-order logic theories represented by sets of clauses. A satisfiable theory, saturated by superposition, implicitly defines a minimal term-generated model for the theory. Unfortunately, checking consistency of existential properties with respect to a saturated theory directly leads to a modification of the Herbrand domain, as new Skolem functions are introduced. At the core of the fixed domain reasoning algorithm of SPASS-FD thus lies a superposition calculus that avoids Skolemization by using an explicit representation of existential variables [9]. To this end, clauses are extended by constraints that restrict the instantiations of the existential variables for a clause. For example, a formula $\exists u.\forall y.P(u,y) \wedge \neg P(0,0)$ corresponds to constrained clauses $u{\simeq}x \parallel \rightarrow P(x,y)$ and $u{\simeq}x \parallel P(a,a) \rightarrow$. Resolving these two clauses and unifying their constraints leads to an empty clause with constraint $u{\simeq}a$, signifying that $u \mapsto a$ does not satisfy the initial clauses. In this setting, a contradiction is formed by a set of constrained empty clauses that together exclude all possible ground instantiations of the existential variables. This property, called *coverage*, can be expressed as unsatisfiability of an equational formula over the empty theory and can hence be decided by disunification.

Analogous to the first-order case, a saturated clause set is not covering iff it has a Herbrand model [9]. When the initial clause set is completed, the only remaining Herbrand model is the minimal model. Hence techniques for reasoning about fixed domain properties can be also used for minimal model reasoning. This turns the previously mentioned constrained superposition calculus into a decision procedure for several classes of minimal model problems [7,8,6].

Reasoning about Equational Literals. Predicate completion as well as minimal model reasoning quickly becomes undecidable in the presence of an equational theory. An important exception are equations of the form $s^n(x){\simeq}s^m(x)$.

Interpretations where such equations hold are called *ultimately periodic*. They occur e.g. as the models of predicative linear time linear logic. In [6], I extended the aforementioned algorithms to encompass ultimate periodicity equations. These extensions are also integrated in SPASS-FD.

It is furthermore possible to allow arbitrary disequations in constraints. This makes it possible to write completions in such a way that the non-constraint part of every clause is predicative [7].

3 Implementation

The tool SPASS-FD is an implementation of the aforementioned algorithms on top of SPASS [14] and, as SPASS, is written in plain C. The main advantage of using SPASS as a basis is that this reasoner is specialized on saturation-based theorem proving and already places efficiently implemented data types at the disposal for its modules. The additions made by SPASS-FD are mostly orthogonal to the other SPASS modules and can be used in conjunction with them. They are activated by the command line option `-PComp=1`.

Because the current version of SPASS does not yet support constraints, a constrained clause $\vec{v}{\simeq}\vec{t}, s_1{\not\simeq}s'_1, \dots, s_n{\not\simeq}s'_n \parallel \Gamma \rightarrow \Delta$ is modeled internally by a regular clause of the form $\mathrm{ExVars}(\vec{t}), \mathrm{CDis}(s_1, s'_1), \dots, \mathrm{CDis}(s_n, s'_n), \Gamma \rightarrow \Delta$.

For easy reference, a graphical overview of the general structure of SPASS-FD is presented in the appendix.

Preprocessing. SPASS-FD takes as its input a problem description and a query of the form $\exists^*\forall^*\phi$. The input (except the query) is as usual processed by FLOTTER, the clause normal form generator of SPASS, and the ultimate periodicity information is extracted. The input clauses are then partitioned with respect to the predicate of their maximal succedent atom. For each such predicate P, a formula ϕ_P is created that describes which ground instances of P hold. Disunification is then used to compute a normalization of $\neg\phi_P$, from which the completion clauses are extracted in a straightforward way and added to the input.

In parallel, existential variables in the query formula are replaced by ExVars literals and the query is then also processed by FLOTTER. The input clauses, their completion and the transformed query are then handed on to saturation.

Saturation. The actual saturation relies completely on the given machinery of SPASS. To deter literals with the predicates ExVars and CDis from interfering with the saturation process, they are artificially kept minimal in the superposition ordering. To avoid the accumulation of multiple ExVars literals, the code of resolution inferences in SPASS has been changed such that ExVars literals are additionally unified during each inference step. This is the only change to the actual saturation machinery of SPASS.

Postprocessing. If saturation terminates, then the resulting constrained empty clauses are checked for coverage, again using disunification. If they are covering, the query $\exists^*\forall^*\phi$ does not hold the minimal model of the input; otherwise a representation of those instances witnessing that it does hold is returned.

4 Optimizations, Testing, and Evaluation

In the superposition component of SPASS-FD, mainly straightforward optimizations have been implemented, like the addition of a clause $\mathrm{CDis}(x, x) \rightarrow$ (corresponding to $x \not\simeq x \parallel \mathrm{false}$) that directly makes all clauses with unsatisfiable constraints redundant. On the other hand, two strong optimizations in SPASS must be deactivated: Splitting changes the minimal model and the especially efficient algorithms for reasoning about sort theories [13] interferes with the semantics of constrained clauses containing ExVars and CDis literals. The latter can be remedied by an explicit integration of constraints into SPASS, which is planned for future releases.

I represent ultimate periodicity equations $s^k(0) \simeq s^l(0)$ using a globally accessible data structure that provides constant-time access to the values k and $l - k$, the constructors s and 0 and the regularly needed terms $s^{k-1}(0)$ and $s^{l-1}(0)$.

Formulas are represented using the term module of SPASS. I adapted the data structures to grant instant access to regularly needed information like parent links and normalization markers that make the non-local changes during normalization (e.g. by replacements like $x \simeq t \wedge \phi \rightsquigarrow x \simeq t \wedge \phi[x \mapsto t]$) tractable. For disunification, which is both code-wise the biggest and computation-wise the most expensive part of the implementation, the nondeterminism of the used algorithm allows for a wide variety of normalization strategies. Motivated by performance increases on the tested examples, I made the following design decisions: Formulas are traversed in a depth search pattern to allow for fast propagation of true or false which often considerably reduces the size of the formula. Formulas are not kept locally in conjunctive normal form, a prerequisite of previous algorithms. Rules that tend to increase formula size (e.g. Sort Reduction, Distribution and Explosion) are only applied to otherwise normalized subformulas.

To the best of my knowledge, there has so far only been one publicly available implementation of disunification [11], which relies on an early inefficient variant of the algorithm and is not maintained any more. The algorithms for disunification and predicate completion over ultimately periodic models and for fixed domain reasoning have not been implemented before. In particular, no benchmarks for these problems exist. Instead, SPASS-FD was tested in the following ways:

- The implementation of the disunification and predicate completion procedures has been tested on and optimized with the help of problems in the TPTP library [12].
- The extension for ultimately periodic interpretations and the overall minimal model reasoning has been tested on hand-crafted problems.
- The decision procedure for contexts has been tested on randomly generated examples.

Collections of problem files are available from the system's homepage.

5 Conclusion

SPASS-FD enriches SPASS by predicate completion and a constrained superposition calculus for existential variables. It is thus the first implementation of saturation-based minimal model reasoning and in particular of all decision procedures from [7] and [6] for queries with a quantifier alternation. Predicate completion is also implemented for a class of non-Horn clause sets; the corresponding theory has been submitted in parallel to this system description.

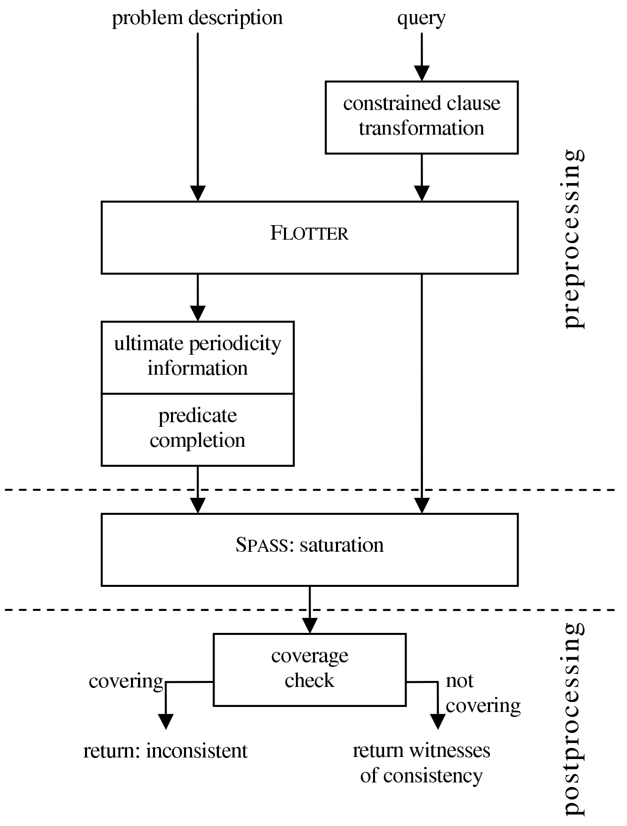
Currently, an extension of constrained superposition to equational clauses [9] is under construction, as are more complicated decision procedures based thereon [8]. A version using proper constraints for full modularity will appear once those are officially supported by SPASS, and the same holds for real multisorting.

References

1. P. Baumgartner and C. Tinelli. The model evolution calculus. In F. Baader, editor, *CADE-19*, volume 2741 of *LNCS*, pages 350–364. Springer, 2003.
2. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322, New York, 1977. Plenum Press.
3. H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3-4):371–425, 1989.
4. H. Comon and R. Nieuwenhuis. Induction = I-axiomatization + first-order consistency. *Information and Computation*, 159(1/2):151–186, May 2000.
5. C. G. Fermüller and R. Pichler. Model representation via contexts and implicit generalizations. In R. Nieuwenhuis, editor, *CADE-20*, volume 3632 of *LNCS*, pages 409–423. Springer, 2005.
6. M. Horbach. Disunification for ultimately periodic interpretations. In E. M. Clarke and A. Voronkov, editors, *LPAR-16*, volume 6355 of *LNAI*, pages 290–311. Springer, 2010.
7. M. Horbach and C. Weidenbach. Decidability results for saturation-based model building. In R. Schmidt, editor, *CADE-22*, volume 5663 of *LNAI*, pages 404–420. Springer, 2009.
8. M. Horbach and C. Weidenbach. Deciding the inductive validity of $\forall\exists^*$ queries. In E. Grädel and R. Kahle, editors, *CSL 2009*, volume 5771 of *LNCS*, pages 332–347. Springer, 2009.
9. M. Horbach and C. Weidenbach. Superposition for fixed domains. *ACM Transactions on Computational Logic*, 11(4):27:1–27:35, November 2010.
10. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier, 2001.
11. N. Peltier. System description: An equational constraints solver. In C. Kirchner and H. Kirchner, editors, *CADE-15*, volume 1421 of *LNCS*, pages 119–123. Springer, 1998.
12. G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automomated Reasoning*, 43(4):337–362, 2009.
13. C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *CADE-16*, volume 1632 of *LNAI*, pages 378–382. Springer, 1999.
14. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In R. Schmidt, editor, *CADE-22*, volume 5663 of *LNCS*, pages 140–145. Springer, 2009.

Appendix

A Graphical System Overview



B An Annotated Example Run

Consider the interpretation given by the following atom with exception: $D = \{P(x, s(y))/\{P(s(x'), s(x'))\}\}$. This is an example taken from [7], where also the translation into a clause set is explained in detail.

To show that the formula $\exists x.P(s(x), x)$ holds for D, the following input is given to SPASS-FD:

```
begin_problem(X).
list_of_descriptions.
name({*Atom with Exceptions Example*}).
author({*Matthias Horbach*}).
status(satisfiable).
description({*Proves a DIG Property*}).
end_of_list.
```

```
list_of_symbols.
functions[(s,1),(0,0)].
predicates[(P,2), (Pp,2), (Pn,2)].
end_of_list.

list_of_formulae(axioms).
formula(forall([x,y],Pp(x,s(y)))).
formula(forall([x],Pn(s(x),s(x)))).
formula(forall([x,y],implies(Pp(x,y),or(Pn(x,y),P(x,y))))).
end_of_list.

list_of_formulae(conjectures).
formula(exists([x],P(s(x),x))).
end_of_list.

list_of_settings(SPASS).
{*
set_flag(PComp,1).
set_flag(Select,0).
set_flag(Sorts,0).
set_precedence(P,Pp,Pn).
set_DomPred(P,Pn,Pp).
*}
end_of_list.
end_problem.
```

The `list_of_descriptions` contains some information on the problem. In the following `list_of_symbols`, the signature is fixed (`Pp` and `Pn` denote the positive and exception parts of the description of D, respectively), and the clauses describing D are given in the `list_of_formulae`. Finally, Spass-FD is told in the `list_of_settings` to apply the predicate completion algorithm to the input, disable selection and the special handling of sorts and choose the precedence `P` $\succ$ `Pn` $\succ$ `Pp`. The last setting causes constraint literals to be minimal. The output of Spass-FD when run on this input begins with the following lines:

```
Perfect SPASS: Conjecture prepared.
Perfect SPASS: Input completed.
--------------------------SPASS-START-----------------------------
```

This signals that the conjecture has been transformed into a constrained clause and the input has been completed. Then the completion is output:

```
Input Problem:
1[0:Inp] ||  -> Pp(U,s(V))*.
2[0:Inp] ||  -> Pn(s(U),s(U))*.
3[0:Inp] || ExVars(U) -> P(s(U),U)*.
4[0:Inp] || Pp(U,V) -> P(U,V)* Pn(U,V).
5[0:Inp] || P(U,V)* -> Pp(U,V).
6[0:Inp] || P(U,V)* Pn(U,V) -> .
```

```
7[0:Inp] || Pn(s(U),V)* CDis(V,s(U)) -> .
8[0:Inp] || Pn(0,U)* -> .
9[0:Inp] || Pp(U,0)* -> .
10[0:Inp] || CDis(U,U)* -> .
```

Clause 3 is the conjecture, the clauses 1,2 and 4–9 form the completed description of D and clause 10 is the additional clause $x \not\approx x \parallel \Box$ introduced for efficiency. SPASS-FD then analyses and saturates the input using the adapted inference rules:

```
 This is a first-order Non-Horn problem without equality.
 Axiom clauses: 9 Conjecture clauses: 1
 Inferences: IORe=1 IOFc=1
 Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RFSub=1
             RBSub=1 RCon=1
 Extras    : Input Saturation, No Selection, Full Splitting,
             Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: s > nequal > div > id > CDis > P > Pp > Pn > ExVars > 0
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
8[0:Inp] || Pn(0,U)* -> .
9[0:Inp] || Pp(U,0)* -> .
10[0:Inp] || CDis(U,U)* -> .
1[0:Inp] ||  -> Pp(U,s(V))*.
2[0:Inp] ||  -> Pn(s(U),s(U))*.
3[0:Inp] || ExVars(U) -> P(s(U),U)*.
5[0:Inp] || P(U,V)* -> Pp(U,V).
6[0:Inp] || Pn(U,V) P(U,V)* -> .
4[0:Inp] || Pp(U,V) -> Pn(U,V) P(U,V)*.
7[0:Inp] || CDis(U,s(V)) Pn(s(V),U)* -> .
        Given clause: 8[0:Inp] || Pn(0,U)* -> .
        Given clause: 9[0:Inp] || Pp(U,0)* -> .
        Given clause: 10[0:Inp] || CDis(U,U)* -> .
        Given clause: 1[0:Inp] ||  -> Pp(U,s(V))*.
        Given clause: 2[0:Inp] ||  -> Pn(s(U),s(U))*.
        Given clause: 3[0:Inp] || ExVars(U) -> P(s(U),U)*.
        Given clause: 5[0:Inp] || P(U,V)* -> Pp(U,V).
        Given clause: 11[0:Res:3.1,5.0] || ExVars(U) -> Pp(s(U),U)*.
        Given clause: 12[0:Res:11.1,9.0] || ExVars(0)* -> .
        Given clause: 6[0:Inp] || Pn(U,V) P(U,V)* -> .
        Given clause: 13[0:Res:3.1,6.1] || ExVars(U) Pn(s(U),U)* -> .
        Given clause: 7[0:Inp] || CDis(U,s(V)) Pn(s(V),U)* -> .
        Given clause: 4[0:Inp] || Pp(U,V) -> Pn(U,V) P(U,V)*.
SPASS V 3.5c
SPASS beiseite: Completion found.
```

The only derived constrained empty clause is clause 12: $u{\simeq}0 \parallel \Box$. Consequently, the final coverage check yields that the conjecture holds for all other instantiations of the existential variable:

```
Conjecture holds in the minimal model of the axioms
      for the following instances: (not (equal U (0)))
Problem: dig.dfg
SPASS derived 6 clauses, backtracked 0 clauses, performed 0 splits
      and kept 13 clauses.
SPASS allocated 25242 KBytes.
SPASS spent     0:00:00.16 on the problem.
                0:00:00.02 for the input.
                0:00:00.02 for the FLOTTER CNF translation.
                0:00:00.00 for the input completion.
                0:00:00.00 for inferences.
                0:00:00.00 for the backtracking.
                0:00:00.00 for the reduction.
                0:00:00.00 for the coverage check.
--------------------------SPASS-STOP------------------------------
```

C Reference: The Constrained Superposition Calculus for Fixed Domains from [9]

Equality Resolution:

$$\frac{\alpha \parallel \Gamma, s{\simeq}t \to \Delta}{(\alpha \parallel \Gamma \to \Delta)\sigma}$$

where

(i) $\sigma = \mathrm{mgu}(s, t)$, and

(ii) either $s{\simeq}t$ is selected in the premise
or no literal is selected and $(s{\simeq}t)\sigma$ is maximal in $(\Gamma, s{\simeq}t \to \Delta)\sigma$.

Equality Factoring:

$$\frac{\alpha \parallel \Gamma \to \Delta, s{\simeq}t, s'{\simeq}t'}{(\alpha \parallel \Gamma, t{\simeq}t' \to \Delta, s'{\simeq}t')\sigma}$$

where

(i) $\sigma = \mathrm{mgu}(s, s')$,

(ii) no literal is selected in the premise
and $(s{\simeq}t)\sigma$ is maximal in $(\Gamma \to \Delta, s{\simeq}t, s'{\simeq}t')\sigma$, and

(iii) $t\sigma \not\succeq s\sigma$

Left Superposition:

$$\frac{\alpha_1 \parallel \Gamma_1 \to \Delta_1, l{\simeq}r \quad \alpha_2 \parallel \Gamma_2, s[l']_p{\simeq}t \to \Delta_2}{(\alpha_1, \alpha_2^{\not\simeq} \parallel \Gamma_1, \Gamma_2, s[r]_p{\simeq}t \to \Delta_1, \Delta_2)\sigma_1\sigma_2}$$

where

(i) $\sigma_1 = \mathrm{mgu}(l, l')$, $\sigma_2 = \mathrm{mgu}(\alpha_1^{\simeq}\sigma_1, \alpha_2^{\simeq}\sigma_1)$,

(ii) no literal is selected in the first premise
and $(l{\simeq}r)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l{\simeq}r)\sigma_1\sigma_2$,

(iii) either $s{\simeq}t$ is selected in the second premise
or no literal is selected and $(s{\simeq}t)\sigma_1\sigma_2$ is maximal in $(\Gamma_2 \to \Delta_2, s{\simeq}t)\sigma_1\sigma_2$,
(iv) $r\sigma_1\sigma_2 \not\succeq l\sigma_1\sigma_2$ and $t\sigma_1\sigma_2 \not\succeq s\sigma_1\sigma_2$, and
(v) l' is not a variable.

Right Superposition:

$$\frac{\alpha_1 \parallel \Gamma_1 \to \Delta_1, l{\simeq}r \quad \alpha_2 \parallel \Gamma_2 \to \Delta_2, s[l']_p{\simeq}t}{(\alpha_1, \alpha_2^{\not\simeq} \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2, s[r]_p{\simeq}t)\sigma_1\sigma_2}$$

where
(i) $\sigma_1 = \mathrm{mgu}(l, l')$, $\sigma_2 = \mathrm{mgu}(\alpha_1^{\simeq}\sigma_1, \alpha_2^{\simeq}\sigma_1)$,
(ii) no literal is selected in any of the premises
and $(l{\simeq}r)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l{\simeq}r)\sigma_1\sigma_2$
and $(s{\simeq}t)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma_2 \to \Delta_2, s{\simeq}t)\sigma_1\sigma_2$,
(iii) $r\sigma_1\sigma_2 \not\succeq l\sigma_1\sigma_2$ and $t\sigma_1\sigma_2 \not\succeq s\sigma_1\sigma_2$, and
(iv) l' is not a variable.

Constraint Superposition:

$$\frac{\alpha_1 \parallel \Gamma_1 \to \Delta_1, l{\simeq}r \quad \alpha_2[l']_p \parallel \Gamma_2 \to \Delta_2}{(\alpha_2[r]_p, \alpha_1^{\not\simeq} \parallel \alpha_1^{\simeq}{\simeq}\alpha_2^{\simeq}[r]_p, \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma}$$

where
(i) $\sigma = \mathrm{mgu}(l, l')$,
(ii) no literal is selected in the first premise
and $(l{\simeq}r)\sigma$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l{\simeq}r)\sigma$,
(iii) $r\sigma \not\succeq l\sigma$, and
(iv) l' is not a variable, and
(v) p is a position in the positive part $\alpha_2^{\simeq}$ of α_2.

Equality Elimination:

$$\frac{\alpha_1 \parallel \Gamma \to \Delta, l{\simeq}r \quad \alpha_2[r']_p \parallel \square}{(\alpha_1, \alpha_2^{\not\simeq} \parallel \Gamma \to \Delta)\sigma_1\sigma_2}$$

where
(i) $\sigma_1 = \mathrm{mgu}(r, r')$, $\sigma_2 = \mathrm{mgu}(\alpha_1^{\simeq}\sigma_1, \alpha_2^{\simeq}[l]_p\sigma_1)$,
(ii) no literal is selected in the first premise
and $(l{\simeq}r)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma \to \Delta, l{\simeq}r)\sigma_1\sigma_2$,
(iii) $r\sigma_1\sigma_2 \not\succeq l\sigma_1\sigma_2$,
(iv) r' is not a variable, and
(v) p is a position in the positive part $\alpha_2^{\simeq}$ of α_2.

D Reference: The Disunification Calculus from [6]

Normalization Rules

Formulas are always kept normalized with respect to these rules.

Propagation of Negation:

$$\begin{array}{llll} \neg\top \rightarrowtriangle \bot & \neg(\phi \vee \phi') \rightarrowtriangle \neg\phi \wedge \neg\phi' & \neg(\exists x.\phi) \rightarrowtriangle \forall x.\neg\phi & \\ & & & \neg\neg\phi \rightarrowtriangle \phi \\ \neg\bot \rightarrowtriangle \top & \neg(\phi \wedge \phi') \rightarrowtriangle \neg\phi \vee \neg\phi' & \neg(\forall x.\phi) \rightarrowtriangle \exists x.\neg\phi & \end{array}$$

Propagation of Truth and Falsity:

$\top \wedge \phi \rightarrow \phi$ $\quad$ $\top \vee \phi \rightarrow \top$ $\quad$ $\exists x.\top \rightarrow \top$
$\bot \wedge \phi \rightarrow \bot$ $\quad$ $\bot \vee \phi \rightarrow \phi$ $\quad$ $\exists x.\bot \rightarrow \bot$
$\phi \wedge \top \rightarrow \phi$ $\quad$ $\phi \vee \top \rightarrow \top$ $\quad$ $\forall x.\top \rightarrow \top$
$\phi \wedge \bot \rightarrow \bot$ $\quad$ $\phi \vee \bot \rightarrow \bot$ $\quad$ $\forall x.\bot \rightarrow \bot$

Quantifier Accumulation:

P1 $\quad \forall \vec{x}.\phi[\forall \vec{y}.\phi']_p \; \forall \vec{x}, \vec{y}.\phi[\phi']_p$
P2 $\quad \exists \vec{x}.\phi[\exists \vec{y}.\phi']_p \; \exists \vec{x}, \vec{y}.\phi[\phi']_p$
if $\vec{x}$ and $\vec{y}$ are not empty in P1,P2 and there is none of the symbols $\neg, \forall, \exists$ between the two melted quantifiers; if a variable of $\vec{y}$ occurs in $\phi[\top]_p$, it is renamed to avoid capturing.

Rules Applicable for all Sorts

Decomposition, Clash, and Occurrence Check:

D1: $f(u_1, \ldots, u_n) \simeq f(u'_1, \ldots, u'_n) \rightarrow u_1 \simeq u'_1 \wedge \ldots \wedge u_n \simeq u'_n$
D2: $f(u_1, \ldots, u_n) \not\simeq f(u'_1, \ldots, u'_n) \rightarrow u_1 \not\simeq u'_1 \vee \ldots \vee u_n \not\simeq u'_n$
C1: $f(u_1, \ldots, u_m) \simeq g(u'_1, \ldots, u'_n) \rightarrow \bot$ $\quad$ if $f \neq g$
C2: $f(u_1, \ldots, u_m) \not\simeq g(u'_1, \ldots, u'_n) \rightarrow \top$ $\quad$ if $f \neq g$
O1: $t \simeq u[t] \rightarrow \bot$ $\quad$ if $u[t] \neq t$
O2: $t \not\simeq u[t] \rightarrow \top$ $\quad$ if $u[t] \neq t$
if $f(u_1, \ldots, u_n)$, t and $u[t]$ belong to a free sort

Quantifier Elimination:

Q1: $\exists \vec{x}.\phi_1 \vee \phi_2 \rightarrow (\exists \vec{x}.\phi_1) \vee (\exists \vec{x}.\phi_2)$ $\quad$ if $\vec{x} \cap \mathrm{var}(\phi_1, \phi_2) \neq \emptyset$
Q2: $\forall \vec{x}.\phi_1 \wedge \phi_2 \rightarrow (\forall \vec{x}.\phi_1) \wedge (\forall \vec{x}.\phi_2)$ $\quad$ if $\vec{x} \cap \mathrm{var}(\phi_1, \phi_2) \neq \emptyset$
Q3: $\exists \vec{x}, x.\phi \rightarrow \exists \vec{x}.\phi$ $\quad$ if $x \notin \mathrm{var}(\phi)$
Q4: $\forall \vec{x}, x.\phi \rightarrow \forall \vec{x}.\phi$ $\quad$ if $x \notin \mathrm{var}(\phi)$
Q5: $\forall \vec{x}, x.x \not\simeq t \vee \phi \rightarrow \forall \vec{x}.\phi\{x \mapsto t\}$ $\quad$ if $x \notin \mathrm{var}(t)$
Q6: $\exists \vec{x}, x.x \simeq t \wedge \phi \rightarrow \exists \vec{x}.\phi\{x \mapsto t\}$ $\quad$ if $x \notin \mathrm{var}(t)$
Q7: $\forall \vec{z}, \vec{x}.y_1 \simeq t_1 \vee \ldots \vee y_n \simeq t_n \vee \phi \rightarrow \forall \vec{z}.\phi$
Q8: $\exists \vec{z}, \vec{x}.y_1 \not\simeq t_1 \wedge \ldots \wedge y_n \not\simeq t_n \wedge \phi \rightarrow \exists \vec{z}.\phi$
if in Q7 and Q8 $y_i \neq t_i$ and $\mathrm{var}(y_i \dot{\simeq} t_i) \cap \vec{x} \neq \emptyset$ for all i and $\mathrm{var}(\phi) \cap \vec{x} = \emptyset$ and the sorts of all variables in $\vec{x}$ contain infinitely many ground terms (in particular, all t_i are of a free sort).
Q1 and Q2 also require that no redex for P1 or P2 is created.

Finite Sort Reduction:

S1: $\forall \vec{x}, x.\phi \rightarrow \forall \vec{x}.\phi\{x \mapsto t_1\} \wedge \ldots \wedge \phi\{x \mapsto t_n\}$
S2: $\exists \vec{x}, x.\phi \rightarrow \exists \vec{x}.\phi\{x \mapsto t_1\} \vee \ldots \vee \phi\{x \mapsto t_n\}$
if the sort S of x is free and finite and $t_1, \ldots, t_n$ are the finitely many ground terms in S.

Distribution:

N1: $\forall \vec{x}.\phi[\phi_0 \vee (\phi_1 \wedge \phi_2)]_p \rightarrow \forall \vec{x}.\phi[(\phi_0 \vee \phi_1) \wedge (\phi_0 \vee \phi_2)]_p$
N2: $\exists \vec{x}.\phi[\phi_0 \wedge (\phi_1 \vee \phi_2)]_p \rightarrow \exists \vec{x}.\phi[(\phi_0 \wedge \phi_1) \vee (\phi_0 \wedge \phi_2)]_p$
if ϕ_0, ϕ_1, ϕ_2 are quantifier-free, $\mathrm{var}(\phi_1) \cap \vec{x} \neq \emptyset$, ϕ_1 is not a conjunction in N1 and not a disjunction in N2 and does not contain a redex for N1 or N2, and there is no negation and no quantifier in ϕ along the path p.

Explosion:

Ex1: $\exists \vec{x}.\phi \rightarrow \bigvee_{f \in \mathcal{F}'} \exists \vec{x}, \vec{x}_f.y \simeq f(\vec{x}_f) \wedge \phi\{y \mapsto f(\vec{x}_f)\}$
if y is free in ϕ, no other rule except Ex2 can be applied, there is in ϕ a literal $y \dot{\simeq} t$ where t contains a universally quantified variable, and $\vec{x}$ is non-empty or ϕ is of

the form $\phi = \forall \vec{x}'.\phi'$. If y is of sort S, then $\mathcal{F}' \subseteq \mathcal{F}$ is the set of function symbols of sort $S_1, \ldots, S_n \to S$.
Ex2: $\forall \vec{x}.\phi \twoheadrightarrow \bigwedge_{f \in \mathcal{F}'} \forall \vec{x}, \vec{x}_f.y \not\simeq f(\vec{x}_f) \vee \phi\{y \mapsto f(\vec{x}_f)\}$
if y is free in ϕ, no other rule can be applied, there is in ϕ a literal $y \simeq t$ or $y \not\simeq t$ where t contains an existentially quantified variable, and $\vec{x}$ is non-empty. If y is of sort S, then $\mathcal{F}' \subseteq \mathcal{F}$ is the set of function symbols of sort $S_1, \ldots, S_n \to S$.

Specific Rules for Ultimately Periodic Sorts

Periodic Reduction:
PR: $A[s^l(t)]_p \twoheadrightarrow A[s^k(t)]_p$
if A is an atom and $s^l(t)$ belongs to an ultimately periodic sort of type (k, l).

Periodic Decomposition:

$$\text{PD1: } s(t) \simeq s(t') \twoheadrightarrow \begin{cases} t \simeq t' & \text{if } t \text{ and } t' \text{ are ground} \\ t \simeq s^{k-1}(0) \vee t \simeq s^{l-1}(0) & \text{if } t \text{ is not ground} \\ & \text{ and } s(t') = s^k(0) \\ t \simeq t' & \text{if } t \text{ is not ground} \\ & \text{ and } t' \text{ is ground} \\ & \text{ and } s(t') \neq s^k(0) \\ t \simeq t' \vee (t \simeq s^{k-1}(0) \wedge t' \simeq s^{l-1}(0)) & \\ \quad \vee (t \simeq s^{l-1}(0) \wedge t' \simeq s^{k-1}(0)) & \text{if } t \text{ and } t' \text{ are not ground} \end{cases}$$

$$\text{PD2: } s(t) \not\simeq s(t') \twoheadrightarrow \begin{cases} t \not\simeq t' & \text{if } t \text{ and } t' \text{ are ground} \\ t \not\simeq s^{k-1}(0) \wedge t \not\simeq s^{l-1}(0) & \text{if } t \text{ is not ground} \\ & \text{ and } s(t') = s^k(0) \\ t \not\simeq t' & \text{if } t \text{ is not ground} \\ & \text{ and } t' \text{ is ground} \\ & \text{ and } s(t') \neq s^k(0) \\ t \not\simeq t' \wedge (t \not\simeq s^{k-1}(0) \vee t' \not\simeq s^{l-1}(0)) & \\ \quad \wedge (t \not\simeq s^{l-1}(0) \vee t' \not\simeq s^{k-1}(0)) & \text{if } t \text{ and } t' \text{ are not ground} \end{cases}$$

if $s(t)$ belongs to an ultimately periodic sort of type (k, l) and $s(t) \dot{\simeq} s(t')$ is irreducible by PR.
For $k = 0$, the atom $\bot$ replaces $t \simeq s^{k-1}(0)$ and $\top$ replaces $t \not\simeq s^{k-1}(0)$.

Periodic Clash Test:

$$\text{PC1: } s(t) \simeq 0 \twoheadrightarrow \begin{cases} t \simeq s^{l-1}(0) & \text{if } k = 0 \text{ and } t \text{ is not ground} \\ \bot & \text{if } k > 0 \text{ or } t \text{ is ground} \end{cases}$$

$$\text{PC2: } s(t) \not\simeq 0 \twoheadrightarrow \begin{cases} t \not\simeq s^{l-1}(0) & \text{if } k = 0 \text{ and } t \text{ is not ground} \\ \top & \text{if } k > 0 \text{ or } t \text{ is ground} \end{cases}$$

if $s(t)$ belongs to an ultimately periodic sort of type (k, l) and $s(t) \dot{\simeq} 0$ is irreducible by PR.

Periodic Occurrence:

$$\text{PO1: } x \simeq s^n(x) \twoheadrightarrow \begin{cases} x \simeq s^k(0) \vee \ldots \vee x \simeq s^{l-1}(0) & \text{if } l - k \text{ divides } n \\ \bot & \text{if } l - k \text{ does not divide } n \end{cases}$$

$$\text{PO2: } x \not\simeq s^n(x) \twoheadrightarrow \begin{cases} x \not\simeq s^k(0) \wedge \ldots \wedge x \not\simeq s^{l-1}(0) & \text{if } l - k \text{ divides } n \\ \top & \text{if } l - k \text{ does not divide } n \end{cases}$$

if x and $s^n(x)$ belong to an ultimately periodic sort of type (k, l) and $n > 0$.

Periodic Sort Reduction:
PS1: $\forall \vec{x}, x.\phi \twoheadrightarrow \forall \vec{x}.\phi\{x \mapsto 0\} \wedge \ldots \wedge \phi\{x \mapsto s^{l-1}(0)\}$
PS2: $\exists \vec{x}.x.\phi \twoheadrightarrow \exists \vec{x}.\phi\{x \mapsto 0\} \vee \ldots \vee \phi\{x \mapsto s^{l-1}(0)\}$
if x belongs to an ultimately periodic sort of type (k, l) and x occurs in ϕ.