Certified Semantics for Disequality Constraints

Dmitry Rozplokhas
Higher School of Economics and JetBrains Research, Russia
rozplokhas@edu.hse.ru

Dmitry Boulytchev
Saint Petersburg State University and JetBrains Research, Russia
dboulytchev@math.spbu.ru

We present a certified semantics for disequality constraints in MINIKANREN. In its initial form [4, 5] MINIKANREN introduces a single form of constraint — unification of finite terms. While from a theoretical standpoint unification together with other primitive constructs (conjunction, disjunction and fresh variable introduction) form a Turing-complete basis, in practice of relational programming a number of extensions are oftenly used to make specifications more expressive, concise or efficient. One of the most important extensions is *disequality constraint*.

Disequality constraint [3] introduces one additional type of base goal — a disequality of two terms

$$t_1 \not\equiv t_2$$

The informal semantics of disequality constraint is complementary to that of unification: it puts certain restrictions on free variables in the terms which prevent them from turning into syntactically equal. Similarly to unification, whose evaluation results in a substitution, which is then threaded through the rest of computations, the effect of disequalify constraint is recorded in a *constraint store* which is later used to check the violation of disequalities [1].

Disequality constraints provide an alternative to the enumeration of finite or infinite domains of terms (depending on a specific program and MINIKANREN version). They are very useful for a special form of case analysis when we should behave in one of two ways depending on whether or not a variable has some specific value and this extension proved to be crucial for problems like constructing relational interpreters in MINIKANREN [2]. Disequality constraints also allow encoding a set of solutions more concisely since the extension always goes together with the ability to present negative information in answers (for example, for a query $(\texttt{x} \not\equiv \texttt{Apple} \wedge \texttt{x} \not\equiv \texttt{Banana})$ MINIKANREN will give a single solution $[\alpha \texttt{ where } \alpha \neq \mathit{Apple}, \alpha \neq \mathit{Banana}]$ with a free variable α instead of the rest of the domain as distinct solutions). At the same time the usage of the negative information by the search is limited for finite domains and if, for example, the domain for x is just Apple and Banana, this solution will still be produced and the search will not halt.

We present an extension of our prior work on certified semantics for core MINIKANREN [6]. In that work we defined denotational semantics, similar to the least Herbrand model, and operational semantics, corresponding to conventional for MINIKANREN *interleaving search*, and proved the soundness and completeness of the latter w.r.t. the former. The development was formally certified in COQ proof assistant, and a correct-by-construction interpreter was extracted.

The contribution of our current work is as follows:

- we extend our denotational semantics to handle disequality constraints;
- we introduce a new abstraction layer (a constraint store with a number of abstract operations) in our operational semantics;

Submitted to:
TEASE-LP 2020

- we formulate a set of *sufficient conditions for completeness*, expressed as algebraic properties of constraint store and abstract operators, and prove the soundness and completeness of operational semantics w.r.t. the denotational one;
- we present a concrete implementation of constraint store and abstract operators and show that they satisfy the sufficient conditions; thus, the soundness and completeness of the implementation with disequality constraints follow immediately, and correct-by-construction interpreter for MINIKANREN with disequalify constraints can be extracted.

The extension of denotational semantics is straightforward (as disequality constraint is complementary to unification). In operational case, we assume that we are given a set of constraint store objects, which we denote by Ω_σ (indexing every constraint store with some substitution σ and assuming the store and the substitution are consistent with each other), and three following operations:

1. Initial constraint store $\Omega_\varepsilon^{init}$ (where ε is empty substitution), which does not contain any constraints yet.
2. Adding a disequality constraint to a store $\mathbf{add}\,(\Omega_\sigma, t_1 \not\equiv t_2)$, which may result in a new constraint store Ω'_σ or a failure $\bot$, if the new constraint store is inconsistent with the substitution σ.
3. Updating a substitution in a constraint store $\mathbf{update}\,(\Omega_\sigma, \delta)$ to intergate a new substitution δ into the current one, which may result in a new constraint store $\Omega'_{\sigma\delta}$ or a failure $\bot$, if the constraint store is inconsistent with the new substitution.

The definition of operational semantics for the language with disequality constraints is now straightforward: for unification we use **update** operation and for disequality constraint we use **add**. In both cases the search in the current branch is pruned if these primitives return $\bot$.

To prove the soundness and completeness result we need a mean to relate both denotational and operational semantics. As in our prior work, this can be done by prescribing a denotational interpretation (denoted by "$[\![\bullet]\!]$") not only to goals, but also to substitutions and constraint stores. Thus, we may formulate the following set of *sufficient conditions* for soundness and completeness:

1. $[\![\Omega_\varepsilon^{init}]\!] = \mathscr{D}$ (where $\mathscr{D}$ is the whole domain in our denotational semantics);
2. $\mathbf{add}\,(\Omega_\sigma, t_1 \not\equiv t_2) = \Omega'_\sigma \implies [\![\Omega'_\sigma]\!] \cap [\![\sigma]\!] = [\![\Omega_\sigma]\!] \cap [\![t_1 \not\equiv t_2]\!] \cap [\![\sigma]\!]$;
3. $\mathbf{add}\,(\Omega_\sigma, t_1 \not\equiv t_2) = \bot \implies [\![\Omega_\sigma]\!] \cap [\![t_1 \not\equiv t_2]\!] \cap [\![\sigma]\!] = \varnothing$;
4. $\mathbf{update}\,(\Omega_\sigma, \delta) = \Omega'_{\sigma\delta} \implies [\![\Omega'_{\sigma\delta}]\!] \cap [\![\sigma\delta]\!] = [\![\Omega_\sigma]\!] \cap [\![\sigma\delta]\!]$;
5. $\mathbf{update}\,(\Omega_\sigma, \delta) = \bot \implies [\![\Omega_\sigma]\!] \cap [\![\sigma\delta]\!] = \varnothing$.

These conditions state that given denotational interpretation and the given operations on constraint stores are adequate to each other. The conditions 2-5 describe exactly what we need to prove the soundness and completeness for base goals (unification and disequality); at the same time, these conditions have relatively simple intuitive meaning in terms of these two operations and are expected to hold naturally in all reasonable implementations of constraint stores.

References

[1] Claire E. Alvis, Jeremiah J. Willcock, Kyle M. Carter, William E. Byrd & Daniel P. Friedman (2011): *cKanren: miniKanren with Constraints*. In: *Proceedings of the 2011 Annual Workshop on Scheme and Functional Programming*.

[2] William E. Byrd, A. Ballantyne, Gregory Rosenblatt & Matthew Might (2017): *A unified approach to solving seven programming problems (functional pearl)*. Proc. ACM Program. Lang. 1(ICFP), pp. 8:1–8:26.

[3] Hubert Comon (1991): *Disunification: A Survey*. In: Computational Logic - Essays in Honor of Alan Robinson, pp. 322–359.

[4] Daniel P. Friedman, William E. Byrd & Oleg Kiselyov (2005): *The reasoned schemer*. MIT Press.

[5] Jason Hemann & Daniel P. Friedman (2013): *μKanren: A Minimal Functional Core for Relational Programming*. In: Proceedings of the 2013 Annual Workshop on Scheme and Functional Programming.

[6] Dmitry Rozplokhas, Andrey Vyatkin & Dmitry Boulytchev (2019): *Certified Semantics for miniKanren*. In: miniKanren and Relational Programming Workshop, pp. 5:1–5:19.