Program Transformations for miniKanren

No Author Given

No Institute Given

1 Conjunctive Partial Deduction for MiniKanren

The conjunctive partial deduction is performed in two steps: driving and residualization. The driving step constructs a directed graph, which resembles a partial SLDNF tree for Prolog programs. Residual program is constructed from this graph in the straightest way possible.

The graph is almost a tree in the sense that cycles are only possible in one case: the goal in the current node is a renaming of a goal, encountered before. The graph is referred to as the driving tree henceforth.

The driving tree is constructed in the process of driving a goal. Each node contains a current goal and a current substitution, and is closely related to the evaluation step, which produced it.

There are several types of nodes in the driving tree:

1. *Fail* is a leaf node. It corresponds to the refutation of some goal.
2. *Success* is a leaf node. It contains a computed answer for the goal.
3. *Or* node means, that the current goal is a disjunction $g_1 \vee g_2$. The children nodes are the results of driving of goals g_1 and g_2.
4. *Call* node corresponds to the conjunction of relation calls $g_1 \wedge g_2 \wedge \cdots \wedge g_n$, where $g_i = R_i^k t_1 \dots t_k$. It has a single child node, which is produced by unfolding of every call in the conjunction $g'_1 \wedge g'_2 \wedge \cdots \wedge g'_n$, where $g'_i = unfold\ g_i$.
5. *Rename* node is a leaf node which means there is a node among the ancestors which is a renaming of the current goal. The index of the ancestor node is stored in the current node.
6. *Gen* node corresponds to a result of generalization. If there is an ancestor node with a goal, which is embedded in the current goal, and the goals contain the same number of conjuncts, the msg for them is computed and stored in the *Gen* node.
7. *Split* node means, that the embedded node contains fewer conjuncts, than the current one. In this case, the conjunction is split into two subconjunctions, which are driven independently.

First, a miniKanren goal is partially evaluated to a conjunction of relation calls.

1. If there is a disjunction, the disjuncts are driven separately. Using the distribution law, we can get rid of disjunctions in any goal by generating *Or*-nodes with no information loss.

2. All "fresh" constructs are combined together and lifted to the topmost level in every definition; thus, a relational definition body contains no more, than one top-level fresh construct, which encloses a freshless body. This purely syntactic transformation makes the driving easier, since we do not need to interpret the "fresh" construct in an arbitrary position.
3. Every unification is evaluated into a substitution, which is threaded along with the goal while driving. A fail to unify means refutation and thus a *Fail* node is created.
4. If after these steps a goal is an empty list of conjunctions, a *Success* node is created with the substitution computed.

When the current goal is the conjunction of relational calls, one of the following cases takes place:

1. If the current goal is a renaming of some goal encountered before, then a *Renaming* leaf node is created since there is no need for driving the goal further.
2. If the current goal is not a renaming of some ancestor goal, we check it for embedding of some ancestor goal. For an embedding, one of two options is possible:
   (a) The number of conjuncts is the same, which means the subterms of the "larger" conjunction embed subterms of the "smaller". In this case, the msg is computed and is driven further.
   (b) The current conjunction has more conjuncts, than the ancestor conjunction. In this case, the conjunction is split into two subconjunctions using the *split* function. Let $g_1 \wedge \cdots \wedge g_n \trianglelefteq h_1 \wedge \cdots \wedge h_k, n < k$. In this case, the conjunction $h_1 \wedge \cdots \wedge h_k$ is split into two subconjunctions $h_1 \wedge \cdots \wedge h_j$ and $h_{j+1} \wedge \cdots \wedge h_k$, such that $g_i \trianglelefteq h_i$ for each $i \in [1..j]$ and $g_{j+1} \not\trianglelefteq h_{j+1}$. The two subconjunctions are then driven independently and the results are connected by a *Split* node.
3. If the current goal is neither renaming nor embedding of any ancestor, it is driven further.

Embedding we use currently is the classic one, which ignores the variables by making any variable embed into any other variable: $X \trianglelefteq Y$ for any variables X, Y. The embedding on the lists of conjuncts requires the heads of the conjuncts to be calls of the same relation.

$$\frac{}{X \trianglelefteq Y},\ \textit{variable embedding}$$

$$\frac{\exists i \in [1..n].X \trianglelefteq t_i}{X \trianglelefteq C^n t_1 \ldots t_n},\ \textit{term diving}$$

$$\frac{\forall i \in [1..n].\ t_i \trianglelefteq u_i}{C^n t_1 \ldots t_n \trianglelefteq C^n u_1 \ldots u_n},\ \textit{term coupling}$$

$$\frac{\forall i \in [1..n].\ t_i \trianglelefteq u_i}{R^n t_1 \ldots t_n \trianglelefteq R^n u_1 \ldots u_n},\ \textit{relation call}$$

$$\frac{R_g^n t_1 \dots t_n \trianglelefteq R_h^n u_1 \dots u_n, g_2 \wedge \dots \wedge g_n \trianglelefteq h_2 \wedge \dots \wedge h_n}{R_g^n t_1 \dots t_n \wedge g_2 \wedge \dots \wedge g_n \trianglelefteq R_h^n u_1 \dots u_n \wedge \dots \wedge h_n}, \textit{ conjunction coupling}$$

$$\frac{g_1 \wedge \dots \wedge g_n \trianglelefteq h_2 \wedge \dots \wedge h_n}{g_1 \wedge \dots \wedge g_n \trianglelefteq h_1 \wedge h_2 \wedge \dots \wedge h_n}, \textit{ conjunction diving}$$

You proposed a different embedding before, the one named "new homeomorphic embedding" ($\trianglelefteq^{\wedge}$), which discriminates the variables in such a way, that a variable is only embedded into itself.

However, we have found a program, whose driving fails to terminate with the new embedding. It is `smallesto` — a relation between a list, its smallest element, and the rest of the elements.

In Prolog syntax it can be defined as follows:

```
leo(0,Y,true).
leo(succ(X),0,false).
leo(succ(X),succ(Y),Z) <- leo(X,Y,Z).

gto(0,Y,false).
gto(succ(X),0,true).
gto(succ(X),succ(Y),Z) <- gto(X,Y,Z).

minmaxo(X,Y,X,Y) <- leo(X,Y,true).
minmaxo(X,Y,Y,X) <- gto(X,Y,true).

smallesto([S],S,[]).
smallesto([H|T],S,[M|Tt]) <- minmaxo(H,Ss,S,M), smallesto(T,Ss,Tt).
```

Several branches in the driving tree of `smallesto(x, y, z)` with `x`, `y` and `z` fresh variables have the following pattern:

```
0) smallesto(V0, V1, V2)
1) minmaxo(V3, V5, V1, V7) /\ smallesto(V4, V5, V6)
2) leo(V3, V5, true) /\ minmaxo(V8, V10, V5, V12]) /\ smallesto(V9, V10, V11)
3) gto(V8, V10, true) /\ minmaxo(V19, V21, V10, V23) /\ smallesto(V20, V21, V22)
4) gto(V31,V32, true) /\ leo(succ(V32), V21, true) /\
       minmaxo(V33, V35, V21, V37])) /\ smallesto(V34, V35, V36])
5) ...
```

In this program, the goals grow "to the left" during the driving. Our embedding requires conjunctions to be coupled, which means, that their first conjuncts should be the calls of the same relation. Because of that, only the goal 3) can possibly be embedded into the goal 4). But, since the original embedding relation does not ignore variables, they are not. Indeed, the third argument of `minmaxo` in the goal 3) is `V10` — the same variable as the second in `gto`. Unfortunately, in the goal 4) the second variable of `gto` is `V32`, while the third one of `minmaxo`

is `V21`. In the goal 4) `V21` is also present in the `leo` relation — and this link between the variables makes it impossible for the driving to terminate in this branch.

Another problem is that the *split* only splits a goal into two subgoals, and if each of the relational calls involved unfolds in a conjunction of at least two relation calls, the goals are going to grow faster, than the *split* splits.

If we have the following program:

```
a(X) <- b(X), c(X)
b(X) <- a(X), b(X), c(X)
c(X) <- a(X), b(X), c(X)
```

Driving of `a(X)` goes like this:

```
1)  a(X)
2)  b(X) /\ c(X) -- nothing is embedded into this goal
3)  a(X) /\ b(X) /\ c(X) /\ a(X) /\ b(X) /\ c(X) -- 1) is embedded into 2); splitting
4a) a(X) -- first subconjuction after split, renaming of 1)
4b) b(X) /\ c(X) /\ a(X) /\ b(X) /\ c(X) -- second subconjunction after split
5b) a(X) /\ b(X) /\ c(X) /\ a(X) /\ b(X) /\ c(X) /\ b(X) /\ c(X) /\
         a(X) /\ b(X) /\ c(X) /\ a(X) /\ b(X) /\ c(X)
...
```

This can probably be fixed by some restriction on unfolding of relational calls, but we do not see a good way to do it. Before, we tried to unconditionally unfold only the first relation in the conjunction, and the following conjunctions are unfolded only if the unfolding is deterministic. This approach seems viable, but still quite heuristic, especially if we do not fix a left-to-right execution strategy.

The last problem we encountered is an enormous size of the residualized programs. The biggest problem is that we introduce dozens of definitions of the same relation. There can be a simple relation (as `gto` or `leo`), which gets to a renaming after one unfolding, but it occurs in different branches of the driving tree, and it is residualized multiple times. For a relational sorting program, the size of the residual program is about 10000 lines.

The rest of the text contains the formal description of the syntax and semantics of the language we work with.

2 Syntax

Supplementary syntax categories:

$$\begin{aligned}
\mathcal{X} &= x_1,\, x_2, \ldots && \text{(syntactic variables)}\\
\mathcal{S} &= \sigma_1,\, \sigma_2, \ldots && \text{(semantic variables)}\\
\mathcal{R}^n &= r_1^n, r_2^n, \ldots && \text{(predicate names indexed by the arity n)}
\end{aligned}$$

Terms:

$$\begin{aligned}
\mathcal{T}(\mathcal{V}) &= \mathcal{V} \,|\, C^n\, t_1, \ldots, t_n && \text{(terms parameterized by variable type)}\\
\mathcal{T}_{\mathcal{X}} &= \mathcal{T}(\mathcal{X}) && \text{(terms which contain only syntactic variables)}\\
\mathcal{T}_{\mathcal{S}} &= \mathcal{T}(\mathcal{S}) && \text{(terms which contain only syntactic variables)}
\end{aligned}$$

Goals:

$$\begin{aligned}
\mathcal{G} &= t_1 \equiv t_2 && \text{(unification)}\\
&\mid\ g_1 \wedge g_2 && \text{(conjunction)}\\
&\mid\ g_1 \vee g_2 && \text{(disjunction)}\\
&\mid\ \underline{\texttt{fresh}}\ x\ \ g && \text{(fresh logical variable binder)}\\
&\mid\ \mathcal{R}^n\, t_1, \ldots, t_n && \text{(relations)}
\end{aligned}$$

Definitions:

$$\mathcal{D}^n = \lambda x_1, \ldots, x_n.g \text{ (relation definition)}$$

States:

$$\begin{aligned}
I &= \mathcal{X} \to \mathcal{T}_{\mathcal{S}} && \text{(syntax variables interpretation)}\\
\Sigma &= \mathcal{S} \to \mathcal{T}_{\mathcal{S}} && \text{(substitutions)}\\
\Delta &= 2^{\mathcal{S}} && \text{(the set of used semantic variables)}\\
\mathfrak{S} &= I \times \Sigma \times \Delta && \text{(states)}\\
\Gamma &= \mathcal{R}^n \to \mathcal{D}^n\\
\Gamma &\vdash \mathfrak{s} \xrightarrow{g} \bar{\mathfrak{s}} && \text{(semantics of the goal g)}
\end{aligned}$$

3 Semantics

$$\Gamma \vdash (\iota, \sigma, \delta) \xrightarrow{t_1 \equiv t_2} [],\ mgu\,(\iota\, t_1)\sigma\ (\iota\, t_2)\sigma = \bot$$

$$\Gamma \vdash (\iota, \sigma, \delta) \xrightarrow{t_1 \equiv t_2} (\iota, mgu\,(\iota\, t_1)\sigma\ (\iota\, t_2)\sigma, \delta)$$

$$\frac{\Gamma \vdash (\iota', \sigma, \delta') \xrightarrow{g} \bar{\mathfrak{s}}}{\Gamma \vdash (\iota, \sigma, \delta) \xrightarrow{\texttt{fresh}\ x\ g} \bar{\mathfrak{s}}},\ \iota' = \iota[x \leftarrow s], \delta' = \delta \cup \{x\}, x \notin \delta$$

$$\frac{\Gamma \vdash (\iota', \sigma, \delta) \xrightarrow{g} \bar{\mathfrak{s}}}{\Gamma \vdash (\iota, \sigma, \delta) \xrightarrow{r^n\, t_1 \ldots t_n} \bar{\mathfrak{s}}},\ \iota' = \iota[x_i \leftarrow t_i\iota], \Gamma\, r^n = \lambda x_1 \ldots x_n.g$$

$$\begin{gathered} \square\iota\ :\ \mathcal{T}_{\mathcal{X}} \to \mathcal{T}_{\mathcal{S}} \\ x\iota = \iota(x) \\ \mathcal{C}^n\, t_1 \ldots t_n = \mathcal{C}^n\, (t_1\iota) \ldots (t_n\iota) \end{gathered}$$

$$\frac{\Gamma \vdash \mathfrak{s} \xrightarrow{g_1} \bar{\mathfrak{s}_1} \quad \Gamma \vdash \mathfrak{s} \xrightarrow{g_2} \bar{s_2}}{\Gamma \vdash \mathfrak{s} \xrightarrow{g_1 \vee g_2} \bar{\mathfrak{s}_1} \oplus \bar{\mathfrak{s}_2}}$$

$$\frac{\Gamma \vdash (\iota, \sigma, \delta) \xrightarrow{g_1} \bar{\mathfrak{s}_1} \quad \Gamma \vdash \bar{\mathfrak{s}_1} \xrightarrow{g_2\iota} \bar{\mathfrak{s}_2}}{\Gamma \vdash (\iota, \sigma, \delta) \xrightarrow{g_1 \wedge g_2} \bar{\mathfrak{s}_2}}$$

$$\frac{\Gamma \vdash \mathfrak{s} \xrightarrow{g} \bar{\mathfrak{s}_1} \quad \Gamma \vdash \bar{\mathfrak{s}} \xrightarrow{g} \bar{\mathfrak{s}_2}}{\Gamma \vdash \mathfrak{s} : \bar{\mathfrak{s}} \xrightarrow{g} \bar{\mathfrak{s}_1} \oplus \bar{\mathfrak{s}_2}}$$