Calculating approximations with dfg2dfg

Enno Keen Christoph Weidenbach

23rd July 2002

1 Introduction

The tool dfg2dfg allows the user to calculate some approximation of a clause set. It is named dfg2dfg because it first reads clauses from an input file in DFG syntax [1]. It then calculates some approximation of the clause set depending on command line options. Finally it writes the approximated clause set in DFG syntax to a file.

2 Synopsis

```
dfg2dfg [-horn] [-monadic] [-linear] [-shallow] infile [outfile]
```

If neither `infile` nor `outfile` are given, dfg2dfg reads from standard input and writes to standard output. If one file name is given, it reads from that file and writes the output to standard output. If more than one file name is given, dfg2dfg reads from the first file and writes to the second.

The following sections describe the effects of the command line options. We are using a notation similar to the notation of reduction rules [2]:

$$\frac{\Theta \parallel \Gamma \rightarrow \Delta}{\Psi_1 \parallel \Pi_1 \rightarrow \Lambda_1} \\ \vdots \\ \Psi_n \parallel \Pi_n \rightarrow \Lambda_n$$

Such a rule is applied to a clause set P by selecting a clause $\Theta \parallel \Gamma \rightarrow \Delta$ from P and replacing it by the clauses $\Psi_i \parallel \Pi_i \rightarrow \Lambda_i$. A transformation is calculated by recursively applying the corresponding rule. The calculation stops when the rule isn't applicable to any clause from the clause set.

3 Transforming a clause to a Horn clause

This transformation is enabled with the `-horn` command line option. The rule is

$$\frac{\Theta \parallel \Gamma \rightarrow E_1, \dots, E_n}{\Theta \parallel \Gamma \rightarrow E_1} \\ \vdots \\ \Theta \parallel \Gamma \rightarrow E_n$$

where $n \geq 2$ and $E_1, \dots, E_n$ are equality or non-equality literals.

4 Transformation to monadic literals

The following two transformations are enabled with the `-monadic` command line option. They transform non-monadic non-equality literals into monadic literals. Note that equality literals are not transformed.

4.1 Transformation by term encoding

This transformation is enabled with the option `-monadic`, which is equivalent to `-monadic=1`. It is described by the rule

$$\frac{\Theta \parallel \Gamma \to P(t_1, \ldots, t_n),\ \Delta}{\Theta \parallel \Gamma \to T(p(t_1, \ldots, t_n)),\ \Delta}$$

where p is a new function corresponding to the predicate P and T is a special predicate. All occurrences of P in the clause set are transformed into the same function p. Constraint and antecedent literals are transformed in a similar way. This approximation is equivalence preserving.

4.2 Transformation by projection

This transformation is enabled with the option `-monadic=2`. It is described by the two rules

$$\frac{\Theta \parallel P(t_1, \ldots, t_n),\ \Gamma \to \Delta}{\Theta \parallel P_1(t_1), \ldots, P_n(t_n),\ \Gamma \to \Delta} \qquad \frac{\Theta \parallel \Gamma \to P(t_1, \ldots, t_n),\ \Delta}{\begin{array}{c}\Theta \parallel \Gamma \to P_1(t_1),\ \Delta \\ \vdots \\ \Theta \parallel \Gamma \to P_n(t_n),\ \Delta\end{array}}$$

where $P_1, \ldots, P_n$ are some new predicates. All occurrences of P in the clause set are transformed into the same predicates $P_1, \ldots, P_n$. Constraint literals are transformed similar to antecedent literals.

5 The linear approximation of a clause

A term is called *linear*, if it contains no repeated variables. This transformation generates the *linear approximation* of a clause with monadic literals by replacing a variable x repeated within the succedent by some new variable x'. Note that the transformation isn't applicable to clauses containing equality or non-monadic literals. This transformation is enabled with the `-linear` option. It is described by the rule

$$\frac{\Theta \parallel \Gamma \to A[x]_p,\ B[x]_q,\ \Delta}{\Theta'\sigma,\ \Theta \parallel \Gamma'\sigma,\ \Gamma \to A,\ B[q/x'],\ \Delta}$$

where (i) all literals are monadic, (ii) $A \neq B$ or $p \neq q$, (iii) x' is a new variable, (iv) $\sigma = \{x \to x'\}$, (v) $\Theta' = \{L \in \Theta \mid x \in \text{vars}(L)\}$ and (vi) $\Gamma' = \{L \in \Gamma \mid x \in \text{vars}(L)\}$.

6 The shallow approximation of a clause

We implemented three kinds of transformations with different requirements for the input clause and different output clauses. The transformations are enabled with the `-shallow` option. Note that the rules aren't applicable to non-horn clauses.

6.1 The strict version

This transformation is enabled with the `-shallow` option, which is equivalent to `-shallow=1`. It is described by the rule

$$\frac{\Theta \parallel \Gamma \to P(t[s]_{p_1})}{\begin{array}{c}S(x),\ \Theta_1 \parallel \Gamma_1 \to P(t[p_1, \ldots, p_n/x]) \\ \Theta_2 \parallel \Gamma_2 \to S(s)\end{array}}$$

where (i) all literals are monadic and all terms in Θ, Γ are variables, (ii) s is a complex term at non-top position p in t, (iii) $\text{vars}(s) \cap \text{vars}(P(t[p_1, \ldots, p_n/c])) = \emptyset$, where c is an arbitrary constant and $p_1, \ldots, p_n$ are all positions of s in t, (iv) x is a new variable and S is a new predicate, (v) Θ_1 and Θ_2 are a partition of Θ, with $\Theta_2 = \{L \in \Theta \mid \text{vars}(L) \subseteq \text{vars}(s)\}$ and (vi) Γ_1 and Γ_2 are a partition of Γ, with $\Gamma_2 = \{L \in \Gamma \mid \text{vars}(L) \subseteq \text{vars}(s)\}$. Note that all occurrences of s in t are replaced simultaneously. This transformation is equivalence preserving with respect to the extension of P in the minimal model.

6.2 A more relaxed version

This version uses the same rule as the strict version, but condition (iii) is omitted. This results in an upper approximation of P. This transformation is enabled with the `-shallow=2` option.

6.3 The least restricted version

This transformation is enabled with the `-shallow=3` option. It uses the rule

$$\frac{\Theta \parallel \Gamma \rightarrow P(t[s]_{p_1})}{\begin{array}{c} S(x),\ \Theta \parallel \Gamma \rightarrow P(t[p_1,\ldots,p_n/x]) \\ \Theta \parallel \Gamma \rightarrow S(s) \end{array}}$$

where (i) all literals are monadic, (ii) s is a complex term at non-top position p_1 in t and $p_1, \ldots, p_n$ refer to all positions of s in t and (iii) x is a new variable and S is a new predicate. Note that Θ and Γ may contain non-variable terms and that s and $t[p_1, \ldots, p_n/c]$ (where c is an arbitrary constant) may share variables. In contrast to the other two versions all negative literals are copied into all resulting clauses. This transformation is an upper approximation of P.

7 Combining several transformations

It is possible to combine several of the transformations described above. However, transformations from the sections 4 or 6 aren't combinable with other transformations from the same section, because they have the same goal. But for the other transformations the order of their application becomes important if several transformations are combined. For example a transformation to monadic literals should be applied *before* the linear transformation because the latter requires monadic literals. If the transformations are applied in this order it might be possible to apply the second transformation more often. dfg2dfg therefore applies the transformations in the same order as the order of their description in this paper. That means transformations are applied in the following order:

1. transformation to horn clauses
2. transformation to monadic literals
3. linear transformation
4. shallow transformation.

The shallow transformation is applied last because it has the most preconditions.

References

[1] Reiner Hähnle, Manfred Kerber, and Christoph Weidenbach. Common syntax of the dfg-schwerpunktprogramm “deduktion”. Interner Bericht 10/96, Universität Karlsruhe, Fakultät für Informatik, Germany, 1996. Current version available from `http://spass.mpi-sb.mpg.de/`.

[2] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.