A Flexible Categorial Formalisation of Term Graphs as Directed Hypergraphs

Wolfram Kahl and Yuhang Zhao

McMaster University, Hamilton, Ontario, Canada, {kahl|zhaoy36}@mcmaster.ca

Term graphs underlie important implementation techniques for functional programming languages, and are also used as internal data structures in many other symbolic computation setting, including in code generation back-ends for example in compilers. Nowadays, term graphs are typically considered as *jungles*, a kind of directed hypergraphs introduced for this purpose by Hoffmann and Plump (1991).

As our formalisation setting, we use the dependently-typed programming language and proof assistant Agda (Norell, 2007); Agda permits us to write definitions essentially in the way they are written for mathematical purposes, and prove properties about them, but all function definitions are also directly executable, making this a good environment for correct-by-construction tool development.

A data type for directed hypergraphs with m input ("variable") nodes and n output ("root") nodes can be defined in the following natural-but-naïve way assuming an arity-indexed label type $\mathsf{Label}_0 : \mathbb{N} \to \mathsf{Set}$ to be given:[1]

```
record DHG₀ (m n : ℕ) : Set₁ where
  field Inner : Set                          -- set of inner (non-input) nodes
  Node = Fin m ⊎ Inner                       -- set containing graph input and inner nodes
  field output : Vec Node n                  -- vector of graph output nodes
        Edge   : Set                         -- set of edges
        eOut   : Edge → Inner                          -- edge output node
        eArity : Edge → ℕ                              -- edge arity
        eLabel : (e : Edge) → Label₀ (eArity e)        -- edge label
        eIn    : (e : Edge) → Vec Node (eArity e)      -- edge input nodes
```

Similar definitions with some discussion of the design space have been shown in (Kahl, 2011); there, we also use Setoid instead of Set, to be able to work with user-defined equalities.

This definition is easily extended to one for jungles, by specifying that eOut is an isomorphism.

However, this definition (no matter whether based on Set or Setoid) is not easily adapted to, for example, finite node and edge sets — one could do this via another extension that adds finiteness proofs.

Using this approach to restrict DHG_0s to those having node and edge sets of shape "Fin n" would involve a type-level propositional equality that would be extremely awkward to use.

[1] For $n : \mathbb{N}$, the type Fin n contains exactly the natural numbers less than n.

For all such restrictions, a much more natural approach is to replace Set in the definition of DHG_0 with a parameter. As a first step, we propose to replace the category of Sets and Agda functions with a parameter category — and turn also the input and output arities into objects of the parameter category. When doing that, however, we still cannot easily switch to a concrete parameter that has exactly the types Fin n as objects, because one of the field $\mathsf{eArity} : \mathsf{Edge} \to \mathbb{N}$ which has a result type that cannot be replaced by any Fin n.[2]

We therefore refine the parameter setting from a category to a category embedded in a semigroupoid via a full and faithful functor preserving finite colimits — the category will be understood as a category of finite sets, and objects like $\mathbb{N}$ are accommodated in the semigroupoid. We can restrict the morphisms of the semigroupoid to those functions starting from finite sets, which makes them implementable via finite data structures such as arrays or balanced trees. (A further refinement using *dependent objects* allows us to deal also with the dependently-typed fields eLabel and eIn.)

With the "standard" parameter setting of finite sets embedded in the category *Set*, the resulting categories of directed hypergraphs (respectively jungles) are equivalent to the mathematically-presented finite directed hypergraphs. However, instantiation with data-structure implementation categories becomes easy, and makes all functions defined in the parameterised setting useful as tool components.

Corradini and Gadducci (1999) introduced *gs-monoidal categories* as categorial theory of term graphs (jungles); we have used the parameterised setting to prove in Agda that directed hypergraphs and jungles form gs-monoidal categories, and implemented, still in that categorial setting, the decomposition function and its correctness proof underlying the theorem of Corradini and Gadducci (1999) that term graphs form the free gs-monoidal category. We are working on using this as foundation for a verified implementation of term graph rewriting.

References

A. Corradini, F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7(4):299–331, 1999. ISSN 1572-9095. doi: 10.1023/A:1008647417502.

B. Hoffmann, D. Plump. Implementing term rewriting by jungle evaluation. *Informatique théorique et applications/Theoretical Informatics and Applications*, 25(5):445–472, 1991.

W. Kahl. Dependently-typed formalisation of typed term graphs. In R. Echahed (ed.), *Proc. of 6th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2011*, EPTCS, vol. 48, pp. 38–53, 2011. doi: 10.4204/EPTCS.48.6. `http://eptcs.org/content.cgi?TERMGRAPH2011`.

[2] One could decide that Fin 2^{64} will "always be enough", but for the sake of generality, let us reject this argument.

U. Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Dept. Comp. Sci. and Eng., Chalmers Univ. of Technology, 2007. See also http://wiki.portal.chalmers.se/agda/pmwiki.php.