Minimization

Guido Tack

2004

1 Graphs and Regular Trees

So far, trees are defined formally as "tree domains" ($TDom \subseteq \mathcal{P}(\mathcal{L}(\mathbb{N}))$). Informally, we know that graphs can model some trees.

To reason about properties of trees, we define the following functions:

$sub(T, i) \in TDom \times \mathbb{N} \rightharpoonup TDom := \{l \mid (i :: l) \in T\}$

$sub(T, p) \in TDom \times \mathcal{L}(\mathbb{N}) \rightharpoonup TDom := \{l \mid (p@l) \in T\}$

$Sub(T) \in TDom \to \mathcal{P}(TDom) := \{sub(T, p) \mid p \in \mathcal{L}(\mathbb{N})\}$

These functions compute the n-th direct subtree, the subtree reached on path p, and the set of all subtrees of some tree domain T. This leads us to the central definition:

Definition 1.1 (Regular Tree) *A tree domain T is called regular iff $Sub(T)$ is finite, i.e. there are only finitely many different subtrees.*

To talk about graphs, we need to define them formally:

Definition 1.2 (Graph) *A graph $G = (V, E)$ consists of a set $V \subseteq \mathbb{N}$ of nodes and an edge function $E \in V \times \mathbb{N} \rightharpoonup V$ such that $\forall v \in V\ \exists n \in N : \{m \in \mathbb{N} \mid (v, m) \in Dom(E)\} = \{0, \ldots, n-1\}$. A graph is called finite if its node set V is finite.*

You might know this definition of graphs from automata theory: V is usually called the set of *states*, and E the *transition function*.

Let's connect our definitions of trees and graphs now.

We can define the *extension* of the transition function E:

$. \in V \times \mathcal{L}(\mathbb{N}) \to V \cup \{\perp\}$

Let $m \in \mathbb{N}$, $p \in \mathcal{L}(\mathbb{N})$. Then

$$
\begin{array}{lll}
v.\epsilon & := v & \\
v.(m :: p) & := E(v,m).p & \text{if } (v,m) \in Dom(E) \\
 & \quad \bot & \text{otherwise}
\end{array}
$$

⇒ $v.p$ is the node reached from v on path p

⇒ $\{p \in \mathcal{L}(\mathbb{N}) \mid v.p \neq \bot\} \in TDom$

Definition 1.3 (Closed Node Set) *A set $X \subseteq V$ is called closed iff $\forall v \in X, p \in \mathcal{L}(\mathbb{N})$: $v.p \neq \bot \implies v.p \in X$*

Definition 1.4 (Tree defined by graph node) $\mathcal{T} \in (V \cup \{\bot\}) \to \mathcal{P}(\mathcal{L}(\mathbb{N}))$

$$
\begin{array}{l}
\mathcal{T}(\bot) := \emptyset \\
\mathcal{T}(v) := \{p \in \mathcal{L}(\mathbb{N}) \mid v.p \neq \bot\}
\end{array}
$$

For $U \subseteq V$, let $\mathcal{T}(U) := \{\mathcal{T}(u) \mid u \in U\}$.

Definition 1.5 (Graph Equivalence) *Two graphs $G = (V,E)$ and $G' = (V',E')$ are equivalent iff $\mathcal{T}(V) = \mathcal{T}(V')$.*

Now we can formulate the following

Proposition 1.1 *Every regular tree can be represented by a node of a finite graph.*

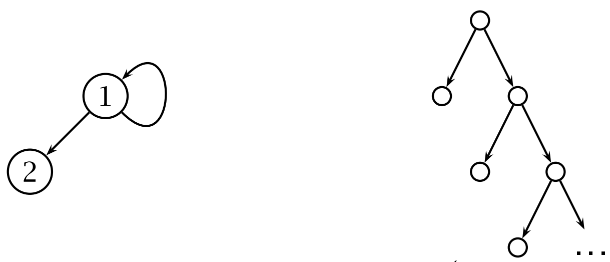
As a proof, we construct a graph $G = (V,E)$ for a regular tree T as follows:

Let $Sub(T) = \{t_1, \ldots, t_n\}$ (finite, as T is regular). Then

$$
\begin{array}{lll}
V & := \{1, \ldots, n\} & \\
E(v,m) & := i & \text{if } sub(t_v, m) = t_i \\
 & \quad \text{undefined} & \text{otherwise}
\end{array}
$$

Then G is a finite graph, and $\mathcal{T}(i) = t_i$ for every i. One of the t_i must be T itself, so this node i in the graph represents the regular tree T.

Example:



The tree on the right is $\mathcal{T}(1)$.

Figure 1: Tree and Graph

2 Relations on Graphs

The canonical and natural relation we can define on graphs now is the following:

$$u \sim_T v \iff \mathcal{T}(u) = \mathcal{T}(v)$$

This relation is an equivalence relation on the nodes of the graph that puts those nodes in the same class which denote the same tree. This gives us a simple

Definition 2.1 (Minimal Graph) *A graph is called minimal iff* $\forall u, v \in V : u = v \iff u \sim_T v$.

Our goal will be to compute $\sim_T$ for an arbitrary graph, and then transform that graph into an equivalent one which is minimal. For this we need to define some more relations:

Definition 2.2 (Congruence) *A congruence on a graph is an equivalence relation* $\sim \in V \times V$ *such that*

$$u \sim v \ \wedge \ u.n \neq \bot \implies v.n \neq \bot \wedge u.n \sim v.n$$

i.e, an equivalence relation that is "compatible" with E. *Another formulation is*

$$u \sim v \implies \mathcal{T}(u) = \mathcal{T}(v).$$

The set of all congruences is called Cong.

The finest congruence is $u \sim v \iff u = v$.

The opposite relation is the following:

Definition 2.3 (Distinction) *A distinction on a graph is an equivalence relation* $\sim \in V \times V$ *such that*

$$\forall \sim' \in Cong : \sim' \subseteq \sim$$

This is equivalent to

$$\mathcal{T}(u) = \mathcal{T}(v) \implies u \sim v.$$

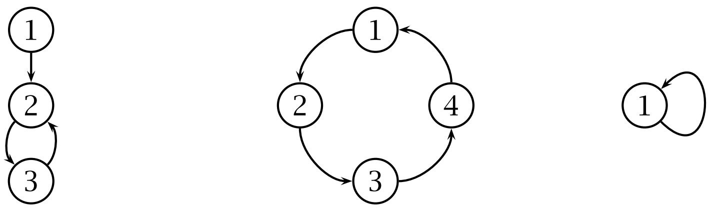
The set of all distinctions is called Dist.

The coarsest distinction is $\sim = V \times V$.

Proposition 2.1 $\sim_T$ *is both the coarsest congruence and the finest distinction.*

3 Partition Refinement

Idea: Take any equivalence relation $\sim$ and compute the coarsest congruence that is a refinement of $\sim$. As an equivalence relation can always be seen as a *partition* of the nodes into equivalence classes, this generic algorithm is called *partition refinement*.



These graphs are all equivalent, the rightmost graph is minimal.

Figure 2: Equivalent graphs

Historically, this is exactly *automaton minimization*, an algorithm developed by Hopcroft [3]. Cardon and Crochemore [1] generalize the idea to arbitrary graphs, and Habib et al. [2] describe a generic and efficient implementation. Mauborgne [4] gives a minimization algorithm that works incrementally. Horbach and Woop [6] give a good formal description of both the original and the incremental algorithm – these lecture notes are based on their work. The author [5] describes how graph minimization can be applied to arbitrary data structures.

3.1 Refinement

Definition 3.1 (Refinement) *We define a function R computing the refinement of a relation on graph nodes:*

$R \in \mathcal{P}(V \times V) \times \mathcal{P}(V) \times \mathbb{N} \to \mathcal{P}(V \times V)$

$R(\sim, X, n) \; := \; (\sim) \cap \{(u, v) \in V \times V \mid u.n \in X \iff v.n \in X\}$

If $\sim, \sim'$ are equivalence relations, we define

$$\sim \succ \sim' \iff \exists X \in V_{/\sim}, n \in \mathbb{N} : \sim' = R(\sim, X, n) \neq \sim$$

Proposition 3.1 (Refinement preserves congruences) *Let $X \in V_{/\sim}$. Then $\sim' \in Cong \wedge \sim' \subseteq \sim \implies \sim' \subseteq R(\sim, X, n)$.*

The proof is left to you as an exercise.

Proposition 3.2 (The fixed point of R is a congruence) $(\forall X \in V_{/\sim}, n \in \mathbb{N} : R(\sim, X, n) = \sim) \implies \sim$ *is a congruence*

The proof is left to you as an exercise.

Corollary 3.1 (Partition refinement computes $\sim_T$) *Let $\sim_0 \succ \sim_1 \succ \cdots \succ \sim_n$ be a chain of distinction relations. Then*

- *this chain is finite.*

- *if there is no $\sim_{n+1}$ such that $\sim_n \succ \sim_{n+1}$, then $\sim = \sim_T$.*

3.2 Runtime

Let k be the maximum arity of all the nodes in V.

Pseudo-code of the generic algorithm:

$i = 0$: *agenda*$_0$ = $(V_{/\sim_0}) \times \{0, \dots, k-1\}$
$i \to i+1$: if *agenda*$_i = \emptyset$ then return $\sim_i$
else $[(X_i, n_i), \dots] =$ *agenda*$_i$
let
$\sim_{i+1} = R(\sim_i, X_i, n_i)$
agenda$_{i+1}$ = updated *agenda*$_i$

The naive algorithm procedes as follows: Every time an equivalence class Y is "split" into Y_1 and Y_2, for every n remove (Y, n) from the agenda and put (Y_1, n) and (Y_2, n) on the agenda. This gives a complexity of $O(n^2)$, where $n = |V|$.

Hopcroft improved this to $O(kn \log n)$, for k the maximum arity of any node in the graph. He noticed that if (Y, n) is not on the agenda, only the smaller one of Y_1 and Y_2 has to be put there.

Both algorithms assume a clever representation of the graph and the equivalence classes to make computation of R efficient. Habib [2] gives a detailed and yet simple description of how to achieve this.

3.3 Minimization

Given a graph $G = (V, E)$ and the relation $\sim_T$ on G, we can easily construct the minimal graph $G' = (V', E')$ that is equivalent to G.

Let $\{[v_1]_{\sim_T}, \dots, [v_n]_{\sim_T}\}$ be the equivalence classes of $\sim_T$.

$V' := \{v_1, \dots, v_n\}$

$E'(v', m) := E(v, m)$ for $v \in [v']_{\sim_T}$

Convince yourself that this is well-defined.

3.4 Labelled Trees and Graphs

Adding labels to trees and graphs is easy. Assuming that we have a set *Labl* of labels, a tree now is a function $t \in$ *Tree* $\subseteq$ *TDom* $\to$ *Lab*. Graphs become tripels: $G = (V, E, L)$, where $L \in V \to$ *Lab*.

The denotation of a graph node has to be adjusted:

$\mathcal{T}(\bot) = \emptyset$

$\mathcal{T}(v)(p) = L(v.p)$ if $v.p \neq \bot$

The canonical equivalence relation of graph nodes has to be aware of labels:

$u \sim_{TL} v \iff L(u) = L(v) \wedge u \sim_T v$

The minimization algorithm doesn't have to be changed at all, only the initial distinction relation must be at least $\sim_L$, defined as $u \sim_L v \iff L(u) = L(v)$ (which is a distinction).

4 Application to Types

Recursive types can be seen as regular, labelled trees with the label set $Lab = \{+, \times, \rightarrow\}$, for example. They can therefore be modelled (and implemented!) as finite graphs.

Computing the minimal graph representing a type has some advantages:

- Type equivalence is decidable in $O(1)$.
- If the type is also needed at runtime, its representation is compact, i.e. memory-efficient.

References

[1] A. Cardon and M. Crochemore. Partitioning a graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science*, 19(1):85–98, July 1982.

[2] M. Habib, Ch. Paul, and L. Viennot. Partition refinement techniques: An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2):147–170, 1999.

[3] J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[4] L. Mauborgne. An incremental unique representation for regular trees. *Nordic Journal of Computing*, 7(4):290–311, 2000.

[5] Guido Tack. Linearisation, minimisation and transformation of data graphs with transients. Diploma thesis, Programming Systems Lab, Universität des Saarlandes, Saarbrücken, May 2003.

[6] S. Woop and M. Horbach. Incremental algorithms and a minimal graph representation for regular trees.
Available from `http://www.ps.uni-sb.de/~horbach/fopra.html`, 2002.