Analysis of Logic Programs Using Regular Tree Languages

Extended Abstract

Gallagher, John Patrick

Published in:
Lecture Notes in Computer Science

Publication date:
2012

Document Version
Early version, also known as pre-print

Citation for published version (APA):
Gallagher, J. P. (2012). Analysis of Logic Programs Using Regular Tree Languages: Extended Abstract. Lecture Notes in Computer Science, 7225, 1-3.

Analysis of logic programs using regular tree languages (extended abstract)

John P. Gallagher*

Roskilde University, Denmark
IMDEA Software Institute, Madrid, Spain
Email: jpg@ruc.dk

Abstract. The field of *finite tree automata* provides fundamental notations and tools for reasoning about set of terms called regular or recognizable tree languages. We consider two kinds of analysis using regular tree languages, applied to logic programs. The first approach is to try to *discover* automatically a tree automaton from a logic program, approximating its minimal Herbrand model. In this case the input for the analysis is a program, and the output is a tree automaton. The second approach is to *expose or check properties* of the program that can be expressed by a given tree automaton. The input to the analysis is a program and a tree automaton, and the output is an abstract model of the program. These two contrasting abstract interpretations can be used in a wide range of analysis and verification problems.

Finite Tree Automata

A tree language is a set of trees, commonly represented as *terms*. Terms are ubiquitous in computing, representing entities as diverse as data structures, computation trees, and computation states. Here we consider only finite terms. Informally, a finite tree automaton (FTA) is a mathematical machine for recognising terms. FTAs provide a means of finitely specifying possibly infinite sets of ground terms, just as finite state automata specify sets of strings. Detailed definitions, algorithms and complexity results can be found in the literature [1]. An FTA includes a grammar for trees given by transition rules of the form $f(q_1, \ldots, q_n) \rightarrow q$, where f is a function symbol from a given signature Σ and $q, q_1, \ldots, q_n$ are states of the automaton. This rules states that a term $f(t_1, \ldots, t_n)$ is accepted by at state q of the FTA if $t_1, \ldots, t_n$ are accepted at states $q_1, \ldots, q_n$ respectively. If q is a *final* state, then the term $f(t_1, \ldots, t_n)$ is recognised by the FTA.

A notational variant of FTAs is given by *alternating tree automata*, though the class of recognisable terms is the same as for FTAs. For the purposes of static analysis, a subset of alternating tree automata we call conjunctive FTA (CFTA) is useful. In a CFTA, transition rules have the form $f(C_1, \ldots, C_n) \rightarrow q$, which is like an FTA transition except that $C_1, \ldots, C_n$ are nonempty sets of automata states. Such a rule states that a term $f(t_1, \ldots, t_n)$ is accepted at state

* Supported by Danish Research Council grants FNU 272-06-0574, FNU 10-084290.

q of the automaton if each t_i is accepted at all the states in C_i. A *bottom-up deterministic* finite tree automaton (DFTA) is one in which the set of transitions contains no two transitions with the same left-hand-side. For every FTA A there exists a DFTA A' that recognises the same set of terms.

An FTA is called *complete* iff for all n-ary functions f and states $q_1, \ldots, q_n$, there exists a state q and a transition $f(q_1, \ldots, q_n) \to q$. This implies that every term is accepted by some (not necessarily final) state of a complete FTA.

Deriving a CFTA from a logic program

The concrete semantics of a definite logic program P is defined by the usual T_P (immediate consequence) operator. We write it as follows, with subsidiary functions $\mathsf{project}$, reduce, ground. Let I be a subset of the Herbrand base of the program.

$$T_P(I) = \{\mathsf{project}(H, \theta\phi)) \mid \begin{array}{l} H \leftarrow B \in P, \theta \in \mathsf{reduce}(B, I), \\ \phi \in \mathsf{ground}(\mathsf{vars}(H) \setminus \mathsf{vars}(B))\} \end{array}$$

The subsidiary functions have obvious meanings in the concrete semantics, yielding the familiar function such that $\mathsf{lfp}(T_P)$ is the least Herbrand model of P. This is the limit of the Kleene sequence $\emptyset, T_P(\emptyset), T_P^2(\emptyset), \ldots$.

We define a finite abstract domain of CFTAs F_P for a given program P, based on the symbols appearing in P, in the style described by Cousot and Cousot [3]. Consider the set of occurrences of non-variable subterms of the heads of clauses in P, including the heads themselves; call this set $\mathsf{headterms}(P)$. The aim is to analyse the instances of the elements of $\mathsf{headterms}(P)$. In particular, the set of instances of the clause heads in successful derivations is the least Herbrand model of P. Let P be a program, with function and predicate symbols Σ. Let Q be a finite set of identifiers labelling the elements of $\mathsf{headterms}(P)$ including a distinguished identifier τ. The base set of transitions Δ_P is the set of CFTA transitions that can be formed from Σ and states Q. The domain F_P is defined as $\mathsf{F}_P = \{\langle Q, \{\tau\}, \Sigma, \Delta\rangle \mid \Delta \subseteq \Delta_P\}$. In other words F_P is the set of CFTAs whose states are identifiers from the finite set Q, have a single accepting state τ and some set of transitions that is a subset of the base transitions of P. Q and Σ are finite, and hence F_P is finite.

Let γ be a concretisation function, where $\gamma(A)$ is the set of terms accepted by CFTA A. The abstract semantic function is a function $T_P^{\alpha_1} : \mathsf{F}_P \to \mathsf{F}_P$ and the safety condition is that $T \circ \gamma \subseteq \gamma \circ T_P^{\alpha_1}$, which is sufficient, together with the monotonicity of T_P, to prove that $\mathsf{lfp}(T_P) \subseteq \gamma(\mathsf{lfp}(T_P^{\alpha_1}))$ [2]. In short, $T_P^{\alpha_1}(A)$ returns a CFTA by solving each clause body wrt A resulting in a substitution of states of A for the body variables. This substitution is projected to the corresponding clause head variables, generating transitions for the returned CFTA.

The sequence $\emptyset, T_P^{\alpha_1}(\emptyset), {T_P^{\alpha_1}}^2(\emptyset), {T_P^{\alpha_1}}^3(\emptyset), \ldots$ is monotonically increasing wrt $\subseteq$. Convergence can be tested simply by checking the subset relation, which is a vital point for efficient implementations (since the language containment check would be very costly). Note that this abstract interpretation is not a Galois connection, in contrast to the second family of analyses discussed next.

Analysis of a program with respect to a given DFTA

A complete DFTA induces a finite partition of the set of accepted terms, since every term is accepted at exactly one state. Given a complete DFTA R, let $\beta_R(t)$ be the partition (denoted by its DFTA state) to which term t belongs. The Herbrand base B_P for a program P is abstracted by mapping into a domain of abstract atoms A_P. More precisely, let $I \subseteq B_P$; define the abstraction function α as $\alpha(I) = \{p(\beta_R(t_1), \ldots, \beta_R(t_n)) \mid p(t_1, \ldots, t_n) \in I\}$. Given a set of abstract atoms $J \subseteq A_P$, the concretisation function γ is defined as $\gamma(J) = \{p(t_1, \ldots, t_n) \mid p(a_1, \ldots, a_n) \in J, t_i$ is accepted at state $a_i, 1 \leq i \leq n\}$. α and γ together define a Galois connection between the complete lattices $(2^{B_P}, \subseteq)$ and $(2^{A_P}, \subseteq)$. There is thus an optimal abstract semantic function $T_P^{\alpha_2} = \alpha \circ T_P \circ \gamma$ [2], and this can be implemented and computed by suitable instantiations of the functions reduce, project and ground mentioned earlier.

Regular types and other applications

Approximating a logic program by a CFTA has been called "descriptive type inference"; each inferred automaton state can be interpreted as a type. It has been used in debugging programs (highlighting inferred empty types) as well as checking whether inferred types correspond to intended types [5]. Analysis based on DFTAs has been used to perform mode analysis, binding time analysis and type checking for given types and modes expressed as DFTAs. Since it is possible to construct a DFTA from any given FTA, this kind of of analysis can be used to check any term property expressible as an FTA. Techniques for handling the potential blow-up caused by conversion to DFTA, as well as BDD representations of the DFTAs to improve scalability, are discussed in [4].

References

1. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.
2. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
3. P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proceedings of the Seventh ACM Conference on Functional Programming Languages and Computer Architecture*, pages 170–181, La Jolla, California, 25–28 June 1995. ACM Press, New York, NY.
4. J. P. Gallagher, K. S. Henriksen, and G. Banda. Techniques for scaling up analyses based on pre-interpretations. In M. Gabbrielli and G. Gupta, editors, *Proceedings of the 21st International Conference on Logic Programming, ICLP'2005*, volume 3668 of *Springer-Verlag Lecture Notes in Computer Science*, pages 280–296, 2005.
5. J. P. Gallagher and G. Puebla. Abstract Interpretation over Non-Deterministic Finite Tree Automata for Set-Based Analysis of Logic Programs. In *Fourth International Symposium on Practical Aspects of Declarative Languages (PADL'02)*, LNCS, January 2002.