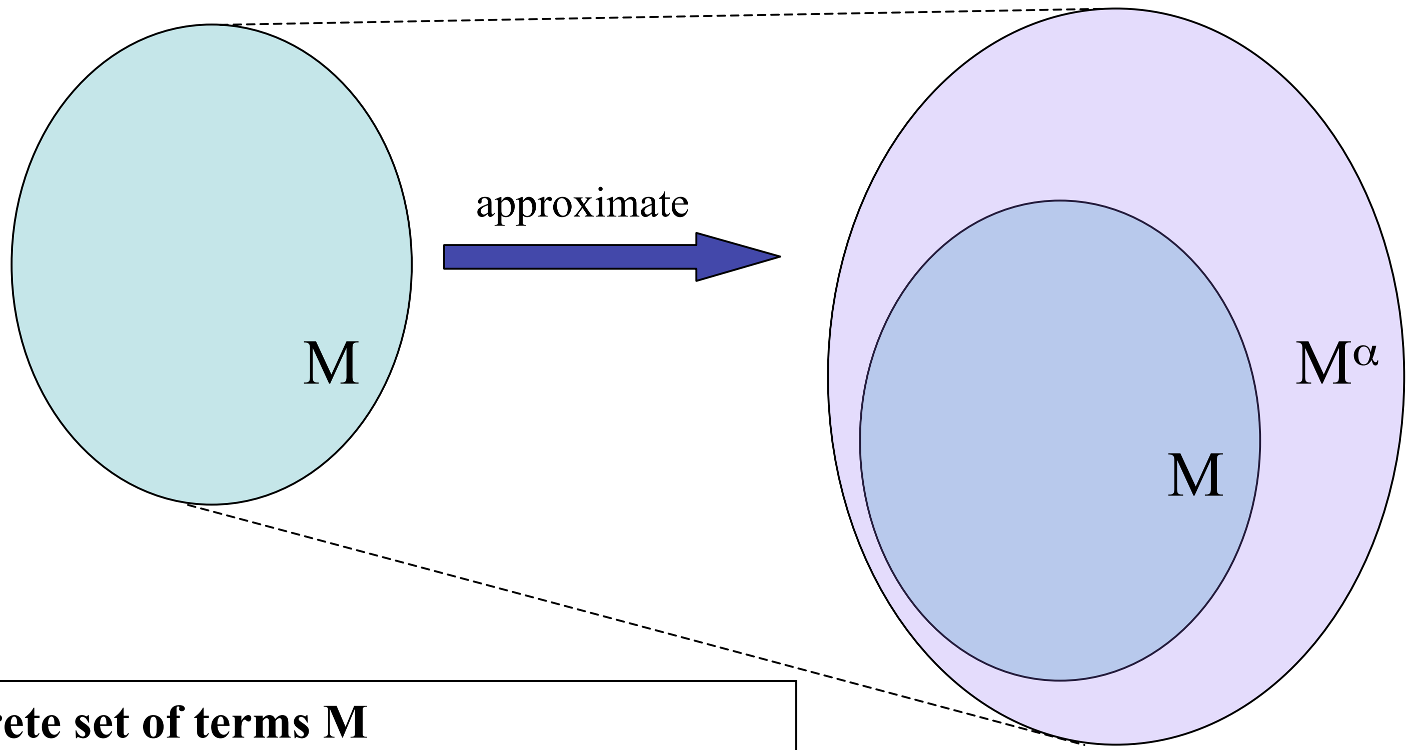
Approximating Term Rewrite Systems: a Horn Clause Specification and its Implementation

John Gallagher   Mads Rosendahl
University of Roskilde, Denmark

Supported by Danish Natural Science Research Council project SAFT

LPAR'2008
Doha, Qatar

Approximating term-based systems



concrete set of terms M

e.g. model of a logic program

e.g. reachable states of a Dolev-Yao model

e.g. reachable terms of a term rewrite system

abstract set of terms M^{α}

$M \subseteq M^{\alpha}$

M^{α} is "easier" to reason about

Proving properties of M in M^α

- Certain properties of M can be proved in an over-approximation M^α.
  - invariants. $\forall x \in M^\alpha.\ p(x) \rightarrow \forall x \in M.\ p(x)$
- A particular kind of invariant
  - safety. badterm $\notin M^\alpha \rightarrow$ badterm $\notin M$

Motivating example using Horn clauses

Horn clauses defining
operations on a token ring (with any
number of processes)
(example from Roychoudury et al,
and Podelski & Charatonik).

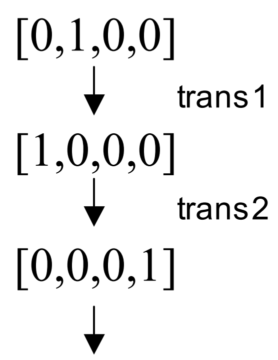
```
init([0,1]).
init([0 | X]) ← init(X).
trans(X,Y) ← trans1(X,Y).
trans([1 |X],[0|Y]) ← trans2(X,Y).
trans1([0,1|T],[1,0 |T]).
trans1([H|T],[H|T1]) ← trans1(T,T1).
trans2([0],[1]).
trans2([H|T],[H|T1]) ← trans2(T,T1).
reachable(X) ← init(X).
reachable(X) ← reachable(Y), trans(Y,X).
```

What are the possible solutions for reachable(X)? Can X be a list containing more than one '1'?

```
init([0,1]).
init([0,0,1) .
init([0,0,0,...,1]).
....
```

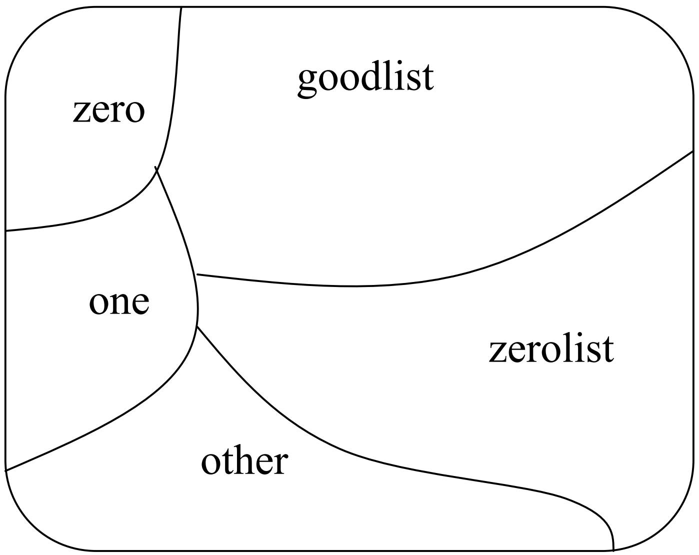
Intended reachable states
reachable([0,0,...,1,...0,0])
(lists with exactly one 1)

Implies mutual exclusion.



Abstract Model

Define a disjoint partition of the set of all terms.



The abstract model shows that only "good" states are reachable, i.e. those containing exactly one "1".

```
% property of interest
0 -> zero.
1 -> one.
[] -> zerolist.
[zero|zerolist] -> zerolist.
[one|zerolist] -> goodlist.
[zero|goodlist] -> goodlist.
```

% abstract model
{reachable(q1),
trans(q1,q1),trans(q3,q3),
trans1(q1,q1),trans1(q3,q3),
trans2(q1,q3),trans2(q2,q1),
trans2(q3,q3)}

Regular Tree Approximations

- Regular tree languages are those definable by finite tree automata (FTAs).

✔ FTAs are a familiar specification language

  ✔ tree grammars

  ✔ abstract syntax

  ✔ regular types

✔ Decision procedures for emptiness, membership

✔ Regular tree languages closed under boolean operations

⇒ Goal - to construct an FTA over-approximating a specified set of terms

⇒ Invariants and safety properties can be decided by FTA operations

Nondeterministic finite tree automata

Example FTA

States
{list, any}
Final States
{list}

Transitions
[] → list
[any | list] → list
[ ] → any
[any | any] → any
c → any

This FTA is nondeterministic.

E.g. [c] is accepted by states list and any.

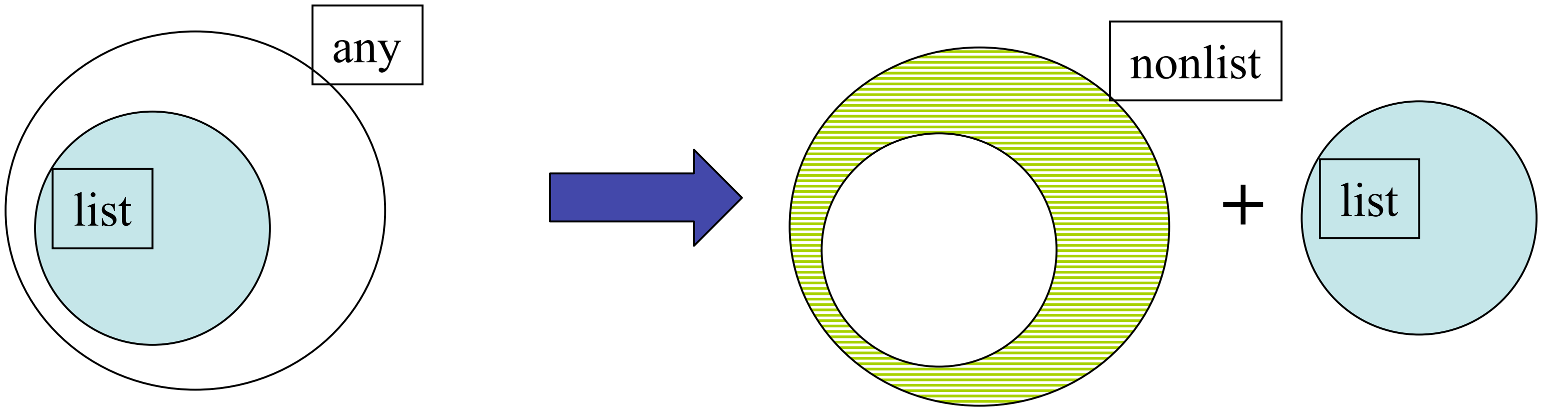
An FTA A defines a set of terms L(A) - the terms that are accepted by some run of A.

Deterministic FTAs

- An FTA is bottom-up deterministic (DFTA) if there are no two rules in Δ having the same left-hand-side.
  - $f(q_1,\ldots,q_n) \rightarrow q$ and $f(q_1,\ldots,q_n) \rightarrow q'$, $q \neq q'$ cannot occur
- For every FTA, there is an equivalent DFTA
- A complete DFTA is one in which there is a transition for every possible lhs.

Determinization of FTAs

- Any FTA can be determinized.
- There is an equivalent FTA that is bottom-up deterministic
- In a deterministic FTA, each term is in at most one type (state). States are disjoint.



Disjoint Accepting States in DFTAs

- In a complete DFTA each term t has exactly one run.
- Hence each term is accepted by one state of a DFTA.

- Thus a complete DFTA defines a disjoint partition.
- The idea is to abstract each term by the (unique) state that accepts it in a DFTA

A procedure for constructing an abstract model of a Horn clause program

- Define an FTA capturing properties of interest
- Determinise the FTA, obtaining a pre-interpretation
- Compute the minimal model wrt to the pre-interpretation
- See [Gallagher & Henriksen 2004] for details

Is it practical?

- Analysis of a program based on an FTA presents two significant practical challenges
  - Determinisation can cause a blow-up in the number of states and transitions
  - Representation and manipulation of relations as tuples is expensive
    - it is like representing Boolean functions using truth tables.

Approaches to Scaling up

- Determinization.
  - Product form of transitions yields much more compact representation of DFTAs
  - Representation of relations. Use a BDD-based representation and exploit techniques from model-checking
  - See [Gallagher, Henriksen & Banda, 2005]

Product representation of transitions

- $f(Q_1,\ldots,Q_n) \to q$ represents the set of transitions

  $\{f(q_1,\ldots,q_n) \to q \mid q_j \in Q_j,\ 1 \leq j \leq n\}$

  E.g. determinized list/nonlist example

  [] → list

  [{list,nonlist}|{list}] → list

  [{list,nonlist}|{nonlist}] → nonlist

  f({list,nonlist},...., {list,nonlist}) → nonlist

Reduction in size with product representation

FTA		DFTA		
Q	Δ	Q_d	(Δ_d)	Δ_Π
3	1933	4	(1130118)	1951
4	1934	5	(10054302)	1951
3	655	4	(20067)	433
4	656	5	(86803)	433
105	803	46	(6567)	141
16	65	16	(268436271)	89

Q = no. of FTA states
Δ = no. of FTA rules
Q_d = no. of DFTA states
Δ_d = no. of DFTA rules
Δ_Π = no. of DFTA product rules

Application to term rewriting

- Problem - Given a set of term rewriting rules and an initial regular set, compute a regular approximation of the reachable terms.

- Many dynamic systems and processes concisely modelled by TRSs
  - cryptographic protocols
  - abstract machines
  - constraint solving procedures
  - equational theories

  ...

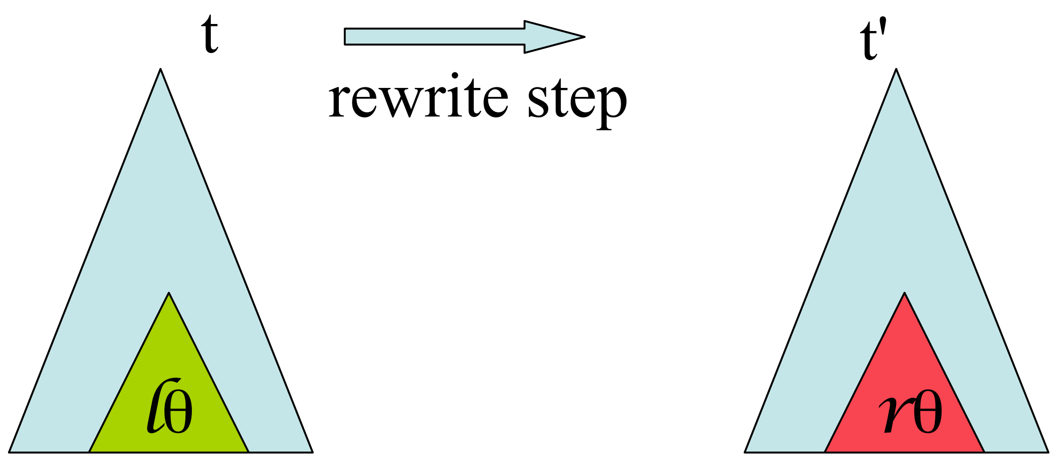
Term rewriting

Signature Σ of ranked function symbols (assumed finite)

Set of variables $\mathcal{V}$

Finite set of rewrite rules $l \Rightarrow r$, where

- l and r are terms constructed from Σ and $\mathcal{V}$
- vars(r) ⊆ vars(l)



Reachable terms

- Write $t \Rightarrow t'$ for a rewrite step
- Write $\Rightarrow^*$ for the reflexive transitive closure of $\Rightarrow$

- Let I be a set of initial terms
- Then a term t is reachable if $t_0 \Rightarrow^* t$ for some $t_0 \in I$.

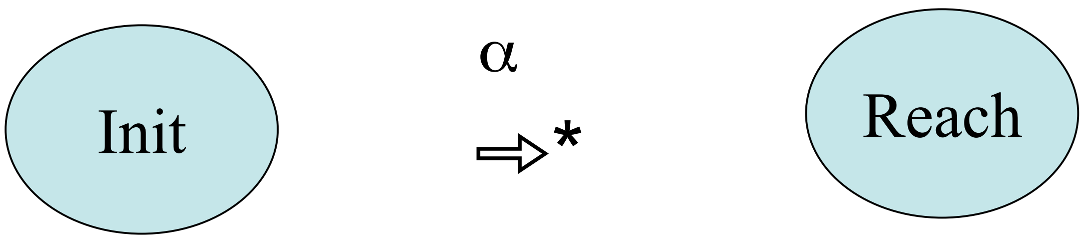
Applications

- Check safety properties
- Optimised compilation (decide statically how a given rule can be applied)
  - limit contexts in which the lhs can appear
  - describe which substitutions are applied to the variables
- Restricting the reachable terms to constructors approximates normal forms
  - debugging

- Note. Rewrite strategy is abstracted away

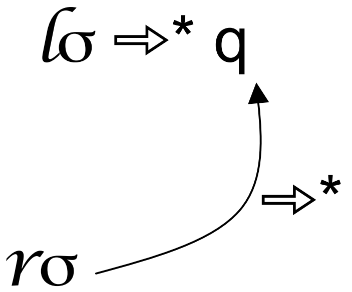
Completion method

- Given a TRS and an initial set specified by an FTA Init
- Compute an FTA Reach containing all the reachable terms (in general a superset)
- Jones & Andersen (1987, 2007) and Feuillade, Genet & Tong (2004) defined a completion method for constructing Reach.



Completion

- Informally - if some state q is reachable from the lhs of a rule FTA, then q must also be reachable from the rhs.



Let A be an FTA
Let σ be a substitution whose domain is the states of A
Let q be a state in A

Add transitions to A to ensure that $r\sigma \Rightarrow^* q$.

New states during completion

- In order to ensure $r\sigma \Rightarrow^* q$, new states need to be added to A.

- Example. $plus(s(X),Y) \Rightarrow s(plus(X,Y))$
  - suppose A contains transitions $s(q_0) \rightarrow q_1$, $plus(q_1,q_2) \rightarrow q_3$. Thus $plus(s(q_0),q_2) \Rightarrow^* q_3$.
  - How to construct a run $s(plus(q_0,q_2)) \Rightarrow^* q_3$?
  - Add a new state, say q_4.
  - Add transitions $plus(q_0,q_2) \rightarrow q_4$, $s(q_4) \rightarrow q_3$.

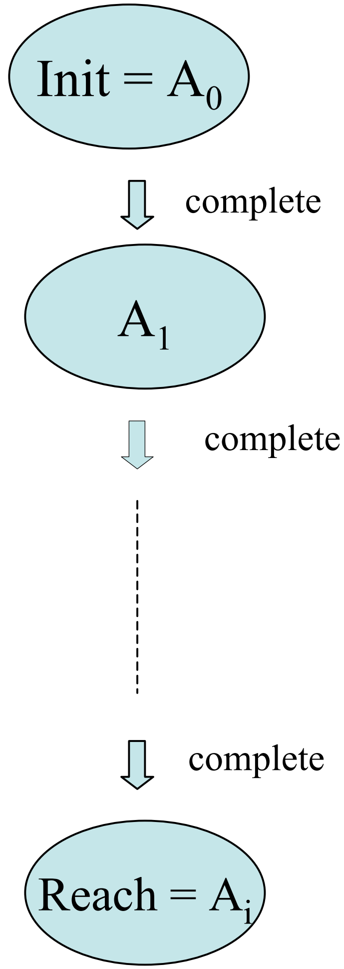
Completion

- Completion algorithm (applies to left-linear TRSs)

Initialise A_0 = Init; i = 0;

repeat
    complete each rule w.r.t. A_i
    add new transitions to
    $A_{i+1} = A_i \cup$ new transitions
    i = i+1
until $A_{i-1} = A_i$

Reach = A_i



Termination of completion procedure

- Termination is not guaranteed
- An infinite number of new states can be introduced $\Rightarrow$ abstraction is required

- Previous work differs in how to avoid infinite number of states
  - Jones & Andersen - fixed finite set of states corresponding to the rhs variables
  - Feuillade et al. - heuristics mapping new states to previous states

The completion step

plus(s(X),Y) ⇒ s(plus(X,Y))

```
plus(A, C)→q0(A,C)) :-
    s(A)→B,
    plus(B,C)→D.
s(A, q0(A,C))→D :-
    s(A)→B,
    plus(B,C)→D.
```

The bodies of the clauses construct a derivation from the lhs of the rule.

The heads of the clauses are the newly introduced transitions.

The term q0(A,C) is the new state introduced.

Complete Example

```
% Example from Feuillade et al. p. 366

plus(0,X) --> X.
plus(s(X),Y) --> s(plus(X,Y)).
even(0) --> true.
even(s(0)) --> false.
even(s(X)) --> odd(X).
odd(0) --> false.
odd(s(0)) --> true.
odd(s(X)) --> even(X).

% initial FTA
even(qpo) -> qf.
even(qpe) -> qf.
s(qeven) -> qodd.
s(qodd) -> qeven.
plus(qodd, qodd) -> qpo.
plus(qeven, qeven) -> qpe.
0 -> qeven.
```

```
rule_odd(B,D) :-
    rule_0(A),
    rule_odd(B,C),
    rule_plus(A,C,D).
rule_false(C) :-
    rule_0(A),
    rule_false(B),
    rule_plus(A,B,C).
rule_true(C) :-
    rule_0(A),
    rule_true(B),
    rule_plus(A,B,C).
rule_even(B,D) :-
    rule_0(A),
    rule_even(B,C),
    rule_plus(A,C,D).
rule_s(B,D) :-
    rule_0(A),
    rule_s(B,C),
    rule_plus(A,C,D).
rule_0(C) :-
    rule_0(A),
    rule_0(B),
    rule_plus(A,B,C).
rule_plus(B,C,E) :-
    rule_0(A),
    rule_plus(B,C,D),
    rule_plus(A,D,E).
rule_plus(A,C,q0(A,C)) :-
    rule_s(A,B),
    rule_plus(B,C,D).
rule_s(q0(A,C),D) :-
    rule_s(A,B),
    rule_plus(B,C,D).
rule_true(B) :-
    rule_0(A),
    rule_even(A,B).
rule_false(C) :-
    rule_0(A),
    rule_s(A,B),
    rule_even(B,C).
rule_odd(A,C) :-
    rule_s(A,B),
    rule_even(B,C).
rule_false(B) :-
    rule_0(A),
    rule_odd(A,B).
rule_true(C) :-
    rule_0(A),
    rule_s(A,B),
    rule_odd(B,C).
rule_even(A,C) :-
    rule_s(A,B),
    rule_odd(B,C).
rule_even(qpo,qf).
rule_even(qpe,qf).
rule_s(qeven,qodd).
rule_s(qodd,qeven).
rule_plus(qodd,qodd,qpo).
rule_plus(qeven,qeven,qpe).
rule_0(qeven).
```

Abstracting the model

- Note. The model of the program is a set of FTA transitions
- If the least model of the program is finite then the FTA in the model approximates the set of reachable terms.
- If infinite, then abstraction techniques for Horn clauses can be applied

Fixed vs. dynamic abstraction

- Relation to abstract interpretation
- Fixed finite height domain
  - (Jones & Andersen's method)
- Infinite height domain with widening
  - (Feuillade et al.'s method)

- Corresponding methods are well-studied in Horn clause model approximation

Initial Experiments

- Literature examples
- Fixed abstractions
  - Jones & Andersen's examples in flow analysis of higher-order functions
- Dynamic abstractions
  - compute an FTA approximating the Horn clause model (both widening-based and other approaches)
  - Use this FTA to define a finite partition
  - evaluate a more precise model using BDD-based evaluation (bddbddb tool).
  - some larger cryptographic protocols, a JVM interpreter (Boichut et al. 2007) have been handled (much faster).

Current Work

- Continued experimental evaluation
- Integrate arithmetic constraints
  - domain of "constrained" tree automata
- Abstraction techniques for TRSs applied to logic programs (effective widenings)
- A more comprehensive Horn clause model for TRSs (allowing for non-linear rules and constraints)
  - modelling the reachable set directly