Non-Termination

Johannes Waldmann, HTWK Leipzig

Why (Non-)Termination

- rewriting models computation
- usually, *termination* is the goal (a computation returns a result $=$ a normal form)
- non-termination means: the rewriting system (program) is “wrong”
- detailed information on non-termination should allow to debug the program
- cf. error messages of a compiler for type errors
- (non-termination is not always bad: infinite data structures, streams, . . .)

Overview of this Talk

Non-Termination of Rewriting:

- easy: looping
- hard: non-looping

Loops—really easy?

- yes: small loops
- no: long loops (*)

note on presentation

- generally, survey style, with examples
- only (*) contains original research

Loops in String Rewriting

rewriting system $R = \{ab \to bbaa\}$,
derivation $\underline{ab}b \to bba\underline{ab} \to bb\boxed{abb}aa$

Defn: a loop is a derivation $u \to_R^+ xuy$
with u, x, y Strings

Thm: If R admits a loop, then R is non-terminating:

$u \to^+ xuy \to^+ xxuyy \to^+ \ldots$

Loops in Term Rewriting

System (SK90/4.54.trs)

$$GF(x,y)) \to F(F(GGx, GGy), F(GGx, GGy))$$

derivation $t = GF(F(p,q), F(GGp, GGq)) \to$
$F(F(GGF(p,q), *), *) \to$
$F(F(\underline{GF(F(GGp, GGq), F(GGp, GGq))}, *), *)$

Context $C[] = F(F(\cdot, *), *)$,
Substitution $\sigma : p \mapsto GGp, q \mapsto GGq$,

Defn: a loop is a derivation $t \to_R^+ C[t\sigma]$
Thm: R looping $\Rightarrow$ R nonterminating.

Non-Looping Non-Termination

Is every non-terminating TRS looping? No.

$$R = \{f(0, y) \to_1 f(y, S0), f(Sx, y) \to_2 f(x, Sy)\}$$

with infinite derivation

$$f(0, 0) \to_1 f(0, S0) \to_1 f(S0, S0) \to_2 f(0, SS0)$$
$$\to_1 f(S^2 0, S0) \to_2 f(S0, S^2 0) \to_2 f(0, S^3 0) \to_1 \ldots$$

Non-Looping Non-Termination

Is every non-terminating SRS looping? No.

idea (Dershowitz, Kurth, Geser and Zantema):

$$ab^nc \to^+ ab^{n+1}c \to^+ \ldots,$$

computed by Turing machine with head moving right (R) or left (L)

$$\{Rb \to bR, Rc \to Lc, bL \to Lb, aL \to abR\}$$

Small Non-Loop. Non-Term. Systems

$\{Rb \to bR, Rc \to Lc, bL \to Lb, aL \to abR\}$
apply ingenious sequence of transformations

- $R \sim b$: $\{bc \to Lc, bL \to Lb, aL \to abb\}$
- $a \sim c$: $\{ba \to La, bL \to Lb, aL \to abb\}$
- introduce additional end markers

$$\{baL \to LaL, bL \to Lb, baL \to babb\}$$

- introduce dummy (X), merge rules

$$\{baL \to LaLXbabb, bL \to Lb\}$$

(Zantema and Geser, 96)

Small Non-Loop. Non-Term. Systems

$$\{baL \to LaLXbabb, bL \to Lb\}$$

is a non-terminating, non-looping

Open: is there such a system with only one rule?
McNaughton's conjecture (1995): No.

related: is termination of *one-rule string rewriting decidable*? (treated by Kurth, Geser, . . .)

(RTALOOP #21, Dauchet): is termination of *one-rule linear (left and right)* TRS decidable?

(Dauchet 1989: for one left-linear rule, no.)

Small Non-Loop. Non-Term. TRS

one-rule non-looping non-terminating
(Zantema and Geser 1996)

$$f(0, Sx, y) \to g(f(0, x, Sy), f(x, y, SS0))$$

One-Rule Non-Loop. Non-Term. TRS

$$f(0, Sx, y) \to g(f(0, x, Sy), f(x, y, SS0))$$

g is *dummy*: $\begin{cases} f(0, Sx, y) \to_1 f(0, x, Sy), \\ f(0, Sx, y) \to_2 f(x, y, SS0) \end{cases}$

second rule only useful for $x = 0$,

gives $f(0, S0, y) \to_2 f(0, y, SS0)$,

write $f(0, x, y) = F(x, y)$ and obtain

$\{F(Sx, y) \to F(x, Sy), F(S0, y) \to F(y, SS0)\}$

with derivations

$F(S0, S^k0) \to_2 F(S^k0, S^20) \to_1^{k-1} F(S0, S^{k+1}0)$

One-Rule Non-Loop. Non-Term. TRS

Is there a *linear* 1-rule non-looping non-terminating TRS? —Perhaps this is (HofWald-6):

$$@(@(0, x), y) \to @(@(x, @(0, y)), 0)$$

write @ as in combinatory logic: $0xy \to x(0y)0$

write $[n+1] = 0[n]$: $\quad (x+1)y = 0xy \to x(y+1)0$

derivation: $0n0 \to n10 \to^{+} 0(n+1)0 \ldots$

TODO: prove absence of loops.

One-Rule Non-Loop. Non-Term. TRS

so far, no automatic proof of non-termination for

$$@(@(0,x),y) \to @(@(x,@(0,y)),0)$$

Remark (Zantema, RTA07):

non-termination is “obvious”, since:

each ground instance of the RHS
contains an instance of the LHS.

Loops

non-looping non-termination is hard . . .

perhaps looping non-termination is easy?

this is indeed decidable:

- input: rewriting system R, number n,
- question: does R admit a loop of length $\leq n$

(proof idea: use Makanin's algorithm for SRS, narrowing $+$ complete case analysis for TRS?)

but . . .

Finding Loops

this is not decidable: input: SRS R, question: is R looping?
it is not even for length-preserving SRS.

proof idea: R is length-preserving $\Rightarrow$ (R is non-terminating $\iff$ R is looping)
termination for lpSRS is undecidable (Caron 1991, Matiyasevich and Senizergues 1996?)

it seems hard already for two rules: e.g. prove termination (or find a loop) for
$\{0000 \to 1011, 1001 \to 0100\}$ (Gebhardt/20)

Finding Loops By Brute Force

(Lankford and Musser 78, Dershowitz 81)
Defn: FC(R) (forward closures) is least set containing R and:

- (inside extension) $(u, xly) \in$ FC$(R) \wedge (l, r) \in R \Rightarrow (u, xry) \in$ FC(R)
- (right extension) $(u, xy) \in$ FC$(R) \wedge (yz, r) \in R \Rightarrow (uz, xr) \in$ FC(R)

Thm: R looping $\iff$ R admits a looping forward closure, i.e. $(u, xuy) \in$ FC(R).

Brute Force (Implementation)

- keep priority queue of closures (pairs of strings)
- initialize with R
- extract smallest, for each successor (from inside extension and right extension):
  - check for loop
  - insert into queue

Brute Force (Implementation II)

important for implementations:

- good hash function (queue represents set)
- good evaluation function (“small” closures first)
  but what is right idea of *size* of (u, v)?
  e.g. $|v|$ or $|u| + |v|$
- handle R and $\mathrm{reverse}(R)$ concurrently

performance example: Match/Jambox find loop in Gebhardt-12 $\{0000 \to 0111, 1011 \to 0010\}$
of length 25, starting with 0000100110,
after enumerating < 1000 closures.

Finding Loops (Variant)

use *overlap* closures (OC), where

- FC: overlaps closure with rules,
- OC: overlaps closure with closures

this is (essentially) the algorithm of NTI (Payet and Mesnard 06)

Finding Loops (Variant)

- Aprove: apply Dependecy Pairs transformation, (restricts set of closures to be enumerated)
- TTT:
  do not enumerate closures until looping one is found,
  instead ask a SAT solver for a looping closure

Long Loops

since existence of loops is undecidable, there must be very long loops. Indeed:

Geser (RTA 02): $\{10^p \to 0^p 1^p 0\}$
has shortest loop of length
$1 + p^0 + p^1 + p^2 + \ldots + p^{p-1}$
(starting from 10^{p^2})

how to (find and) certify long loops, where certificate size (and checking time) is small (e.g. logarithmical in loop length)

Lindenmayer Loops (I)

example: $\{cb \to bba, ab \to bca\}$ (HofWald-1)
We have a *transport system* with *pivot* string b:

$$\forall x \in \Sigma : xb \to b\phi(x)$$
$$\phi : \Sigma \to \Sigma^* : a \mapsto ca, b \mapsto b, c \mapsto ba$$

this implies: $\forall w \in \Sigma^* : wb \to^* b\phi(w)$
e.g. $abc\ b \to ab\ b\ ba = a\ b\ ba \to b\ caba = b\phi(abc)$

and this can be iterated: $\forall k : wb^k \to^* b^k\phi^k(w)$.
e.g. $a\ bbb \to b\ ca\ bb \to^+ bb\ baca\ b \to^+ bbb\ bcabaca$

(iterated morphism: cf. Lindenmayer systems)

Lindenmayer Loops (II)

$$\phi : \Sigma \to \Sigma^* : a \mapsto ca, b \mapsto b, c \mapsto ba$$

$$\begin{aligned}
\phi^1(a) &= ca, \\
\phi^2(a) &= baca, \\
\phi^3(a) &= bcabaca, \\
\phi^4(a) &= bbacabcabaca, \\
\phi^5(a) &= bbcabacabbacabcabaca
\end{aligned}$$

$\phi^5(a) = \ldots a(\ldots b \ldots)^5$ implies the loop:

$$ab^5 \to^+ b^5\phi^5(a) = \ldots a(\ldots b \ldots)^5 \to^+ ab^5$$

Lindenmayer Loops (III)

Geser's example $R_p = \{10^p \to 0^p 1^p 0\}$

admits transport system with
pivot 0^p, morphism $\phi : 1 \mapsto 1^p 0, 0 \mapsto 0$,

it is looping with exponent $p+1$, since
$\phi^2(1) = (1^p 0)^p 0, \ldots$, so $\phi^k(1)$ ends with 10^k, and
$\phi^{p+1}(1) = \phi^p(1^p 0) = (\ldots 10^p)^p$ containing at least
$p+1$ occurences of the pivot 0^p.

Lindenmayer Loops (Implementation)

in Matchbox 2007:

- enumerate (small) overlap closures,
- extract transport systems,
- check whether they are looping
- use blocks of letters

this algorithm is main reason for winning the “non termination” category for string rewriting

Lindenmayer Loops (Example)

size-12-alpha-3-num-385: $\{a \to b, cbab \to aaaccb\}$

pivot aaa, block alphabet $\Gamma = \{a, c, cb\}$

morphism $\phi : \Gamma \to \Gamma^* : a \mapsto a, c \mapsto c\,cb, cb \mapsto c\,cb\,a$

since $caaa \to^2 cbab \to aaaccb$ and
$cbaaa \to cbaba \to aaaccba$

start/exponent: $\phi^7(c)$ contains $c(\ldots a^3 \ldots)^7$

(system has shortest loop of length 21—but it takes much more to find it)

Lindenmayer loops: Open Problem

how to decide the following:

- input: a transport system (pivot p, morphism ϕ)
- question: is it looping: are there a start letter s and an exponent e such that $\phi^e(s) = \ldots s(\ldots p \ldots)^e$

perhaps by growth properties of D0L systems

current implementation tries $e = 1, 2, \ldots$

(applying morphisms is still way better than doing the rewritings since it is deterministic)

Real Life Non Termination

theory is very nice . . .

but will it ever be applied in “real” problems?

Sure - witness the following examples.

Real Life Non Termination Analysis

try compile/execute this Haskell program

```
main = print x where x = 1 + 1 / x
```

Real Life Non Termination Analysis

try to compile this Java program:

```
public class Term {
  public static void main(String[] args) {
    while (true) { }
    System.out.println (42);
  }
}
```