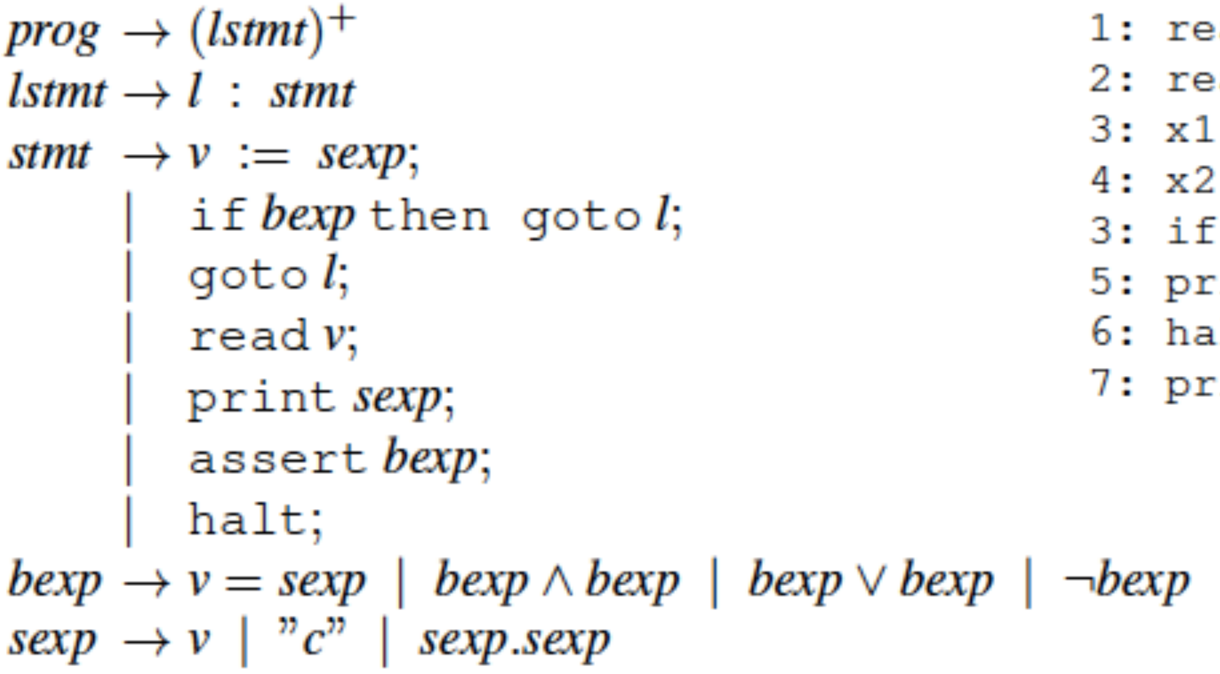
Difficulty of String Analysis, Reachability & Fixpoints

292C

Tevfik Bultan

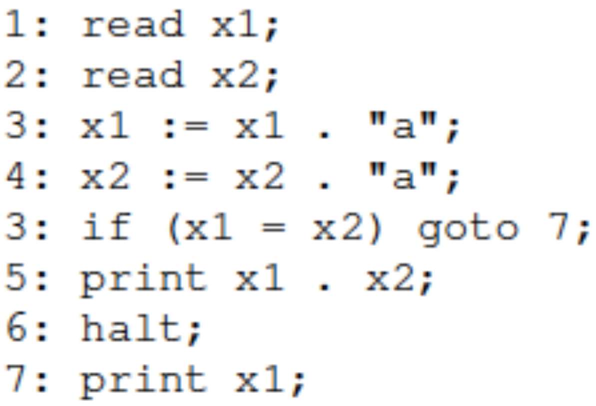
A simple string manipulation language

- Language syntax



$$
\begin{array}{lcl}
\mathit{prog} & \rightarrow & (\mathit{lstmt})^{+} \\
\mathit{lstmt} & \rightarrow & l \;:\; \mathit{stmt} \\
\mathit{stmt} & \rightarrow & v \;:=\; \mathit{sexp}; \\
 & | & \texttt{if}\ \mathit{bexp}\ \texttt{then goto}\ l; \\
 & | & \texttt{goto}\ l; \\
 & | & \texttt{read}\ v; \\
 & | & \texttt{print}\ \mathit{sexp}; \\
 & | & \texttt{assert}\ \mathit{bexp}; \\
 & | & \texttt{halt}; \\
\mathit{bexp} & \rightarrow & v = \mathit{sexp} \mid \mathit{bexp} \wedge \mathit{bexp} \mid \mathit{bexp} \vee \mathit{bexp} \mid \neg \mathit{bexp} \\
\mathit{sexp} & \rightarrow & v \mid \text{"}c\text{"} \mid \mathit{sexp}.\mathit{sexp}
\end{array}
$$

- Example code



```
1: read x1;
2: read x2;
3: x1 := x1 . "a";
4: x2 := x2 . "a";
3: if (x1 = x2) goto 7;
5: print x1 . x2;
6: halt;
7: print x1;
```

Reachability problem

- Reachability problem in string programs:
  - Given a string program P and a program state s
    - where a program state s is defined with the instruction label of an instruction in the program and the values of all the variables,
    - determine if at some point during the execution of the program P, the program state s will be reached.

- Reachability problem for string programs is undecidable (even if we allow only 3 string variables)

Counter machines

- Counter machines are a simple and powerful computational model that can simulate Turing Machines.
- A counter machine consists of a finite number of counters (unbounded integer variables) and a finite set of instructions.
- Counter machines have a very small instruction set that includes an increment, a decrement, a conditional branch instruction that tests if a counter value is equal to zero, and a halt instruction.
- The counters can only assume nonnegative values.
- It is well-known that the halting problem for two-counter machines, where both counters are initialized to 0, is undecidable.
- Two counter machines can simulate Turing Machines.

String programs can simulate counter machines

- A string program P with three string variables (X1, X2, X3) can simulate a counter machine M with two counters (C1, C2)

- We will use the lengths of the strings X1, X2 and X3 to simulate the values of the counters C1 and C2

Where

$C1 = |X1| - |X3|$

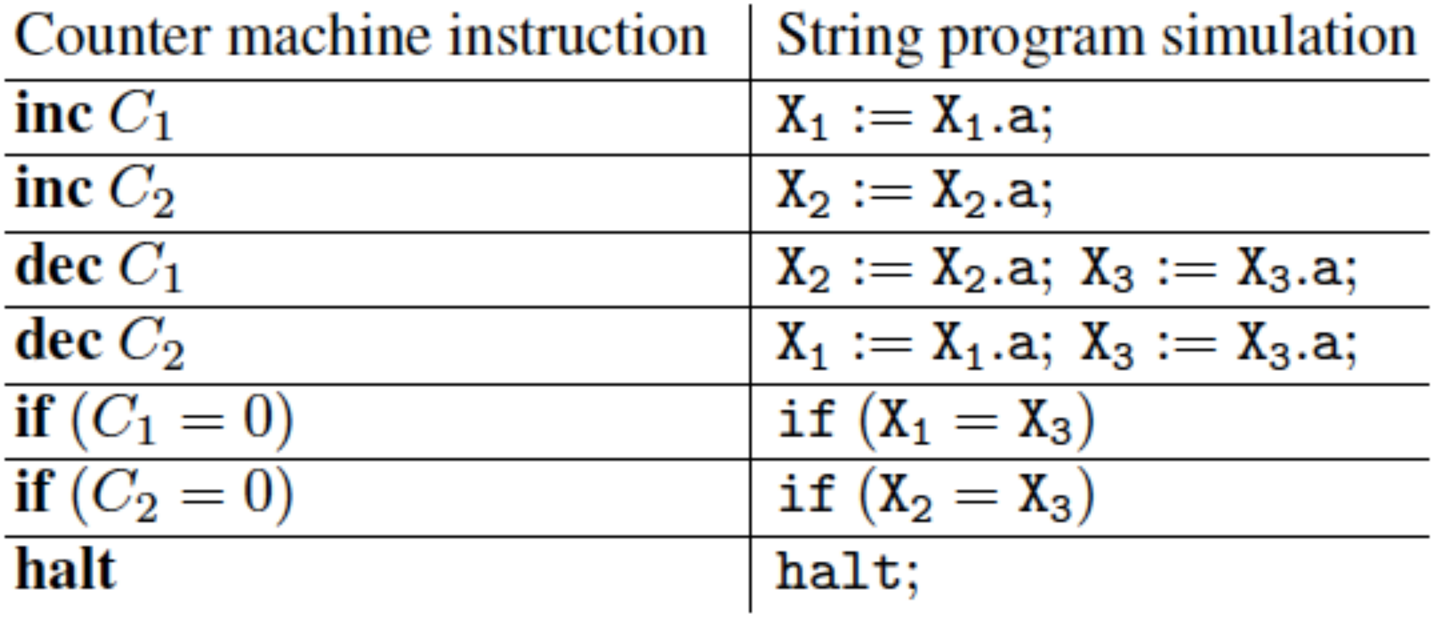
$C2 = |X2| - |X3|$

String programs can simulate counter machines

- M starts from the initial configuration (q0, 0, 0) where q0 denotes the initial instruction and the two integer values represent the initial values of counters C1 and C2, respectively.

- The initial state of the string program P will be (q0, ε, ε, ε) where q0 is the label of the first instruction, and the string variables X1, X2, and X3, are initialized to empty string: ε

Translation of counter-machine instructions to string program instructions



Counter machine instruction	String program simulation
inc C_1	$X_1 := X_1.a;$
inc C_2	$X_2 := X_2.a;$
dec C_1	$X_2 := X_2.a;\ X_3 := X_3.a;$
dec C_2	$X_1 := X_1.a;\ X_3 := X_3.a;$
if $(C_1 = 0)$	$\mathtt{if}\ (X_1 = X_3)$
if $(C_2 = 0)$	$\mathtt{if}\ (X_2 = X_3)$
halt	`halt;`

Reachability problem

- Halting problem for counter machines is undecidable
- String programs can simulate counter machines
- Hence, halting problem for string programs is undecidable.
- Hence, reachability problem for string programs is undecidable.

A richer string manipulating language

$$
\begin{array}{lcl}
prog & \rightarrow & block \\
block & \rightarrow & lstmt^{+} \\
lstmt & \rightarrow & l\ :\ stmt \\
stmt & \rightarrow & v\ :=\ exp; \\
 & | & \texttt{read}\ v; \\
 & | & \texttt{print}\ exp; \\
 & | & \texttt{assert}\ bexp; \\
 & | & \texttt{halt}; \\
 & | & \texttt{if}\ (bexp)\ \texttt{then}\ \{block\} \\
 & | & \texttt{if}\ (bexp)\ \texttt{then}\ \{block\}\ \texttt{else}\ \{block\} \\
 & | & \texttt{while}\ (bexp)\ \{block\} \\
exp & \rightarrow & sexp \mid iexp \\
bexp & \rightarrow & sexp = sexp \\
 & | & \texttt{match}(sexp, sexp) \\
 & | & \texttt{contains}(sexp, sexp) \\
 & | & \texttt{begins}(sexp, sexp) \\
 & | & \texttt{ends}(sexp, sexp) \\
 & | & iexp = iexp \mid iexp < iexp \mid iexp > iexp \\
 & | & bexp \wedge bexp \mid bexp \vee bexp \mid \neg bexp \\
iexp & \rightarrow & v \mid n \mid iexp + iexp \mid iexp - iexp \\
 & | & \texttt{length}(sexp) \\
 & | & \texttt{indexof}(sexp, sexp) \\
sexp & \rightarrow & v \mid \text{``}c\text{''} \mid sexp.sexp \mid sexp^{*} \mid sexp|sexp \\
 & | & \texttt{replace}(sexp, sexp, sexp) \\
 & | & \texttt{substring}(sexp, iexp, iexp) \\
 & | & \texttt{charat}(sexp, iexp) \\
 & | & \texttt{reverse}(sexp)
\end{array}
$$

Semantics

$$\texttt{match}(s, r) \Leftrightarrow s \in \mathcal{L}(r)$$

$$\texttt{contains}(s, t) \Leftrightarrow \exists s_1, s_2 \in \Sigma^* \ : \ s = s_1 t s_2$$

$$\texttt{begins}(s, t) \Leftrightarrow \exists s_1 \in \Sigma^* \ : \ s = t s_1$$

$$\texttt{ends}(s, t) \Leftrightarrow \exists s_1 \in \Sigma^* \ : \ s = s_1 t$$

Semantics

$$t = \mathtt{substring}(s, i, j) \Leftrightarrow \exists s_1, s_2 \in \Sigma^* \; : \; s = s_1 t s_2 \; \wedge \; |s_1| = i \; \wedge \; |t| = j - i$$

$$t = \mathtt{charat(s, i)} \Leftrightarrow \exists s_0, s_1, \ldots, s_n \in \Sigma \; : \; s = s_0 s_1 \ldots s_n \wedge 0 \leq i \leq n \wedge t = s_i$$

$$t = \mathtt{reverse}(s) \Leftrightarrow \exists s_0, s_1, \ldots, s_i \in \Sigma \; : \; s = s_0 s_1 \ldots s_i \; \wedge \; t = s_i \ldots s_1 s_0$$

Semantics

$$(\texttt{length}(s) = 0 \Leftrightarrow s = \epsilon) \land (\texttt{length}(s) = n \Leftrightarrow \exists c_1, c_2, \ldots, c_n \in \Sigma \, : \, s = c_1 c_2 \ldots c_n)$$

$$\begin{aligned}
&(\texttt{indexof}(s,t) = -1 \Leftrightarrow \neg\texttt{contains}(s,t)) \land \\
&(\texttt{indexof}(s,t) = n \quad \Leftrightarrow \quad (\exists s_1, s_2 \in \Sigma^* \; : \; s = s_1 t s_2 \land |s_1| = n) \\
&\qquad\qquad\qquad\qquad\quad \land(\forall i < n \; : \; \neg(\exists s_1, s_2 \in \Sigma^* \; : \; s = s_1 t s_2 \land |s_1| = i)))
\end{aligned}$$

Semantics

$$
\begin{aligned}
& r = \mathtt{replace}(s, p, t) \Leftrightarrow ((\neg\mathtt{contains}(s, p)\ \wedge\ r = s) \vee \\
& (\exists s_3, s_4, s_5 \in \Sigma^* : s = s_3 p s_4\ \wedge\ r = s_3 t s_5\ \wedge\ s_5 = \mathtt{replace}(s_4, p, t) \wedge \\
& (\forall s_6, s_7 \in \Sigma^* : s = s_6 p s_7 \Rightarrow |s_6| \geq |s_3|)))
\end{aligned}
$$

Semantics of a string program

- Semantics of a string program can be defined as a transition system

- A ***transition system*** T = (S, I, R) consists of
  - a set of states S
  - a set of initial states $I \subseteq S$
  - and a transition relation $R \subseteq S \times S$

Semantics of a string program

- Let L denote the labels of program statements, and assume n string and m integer variables, then the set of states of the string program can be defined as:

$$S = L \times (\Sigma^*)^n \times (\mathbb{Z})^m$$

and the initial state is (where *l_1* is the label of the first statement):

$$I = \{\langle l_1, \epsilon, \ldots, \epsilon, 0, \ldots, 0\rangle\}$$

Semantics of a string program

- Given a statement labeled *l*, its transition relation can be defined as a set of tuples:

$$r_l \subseteq S \times S$$

where $(s_1, s_2) \in r_l$

means that executing statement *l* in state s_1 results in state in s_2

- Then, the transition relation of the whole program can be defined as:

$$R = \bigcup_{l \in L} r_l$$

Post condition function

- Using the transition relation, we can define the post condition function that identifies, given a state which state the program will transition.

$$s_2 = \textsc{post}(s_1, l) \Leftrightarrow (s_1, s_2) \in r_l$$

$$s_2 = \textsc{post}(s_1) \Leftrightarrow \exists l \in L \ : \ s_2 = \textsc{post}(s_1, r_l)$$

$$s_2 = \textsc{post}(s_1) \Leftrightarrow (s_1, s_2) \in R$$

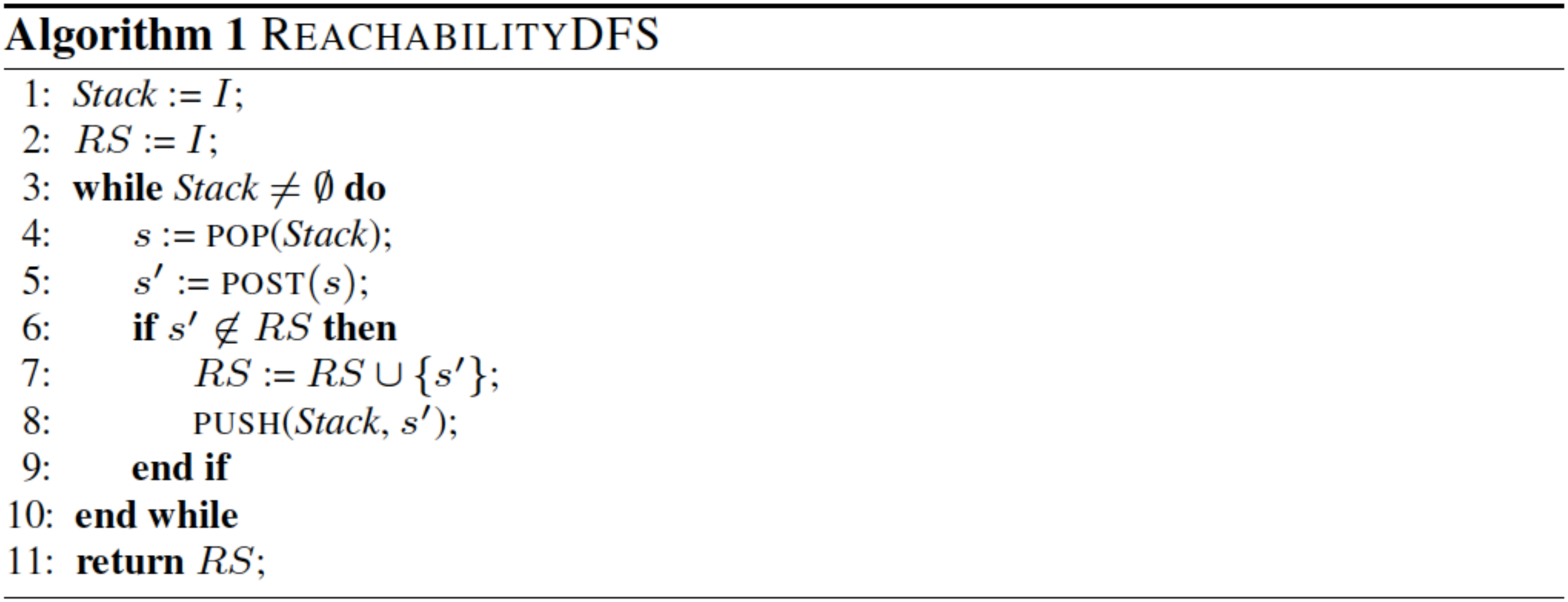
Computing reachable states

- The set of states that are reachable from the initial states of the program can be defined as:

$$RS = \{s \mid \exists s_0, s_1, \ldots, s_n \ : \ \forall i < n \ : \ (s_i, s_{i+1}) \in R \ \wedge \ s_0 \in I \ \wedge \ s_n = s\}$$

- Reachable states can be computed using a simple depth-first-search

Computing reachable states with DFS



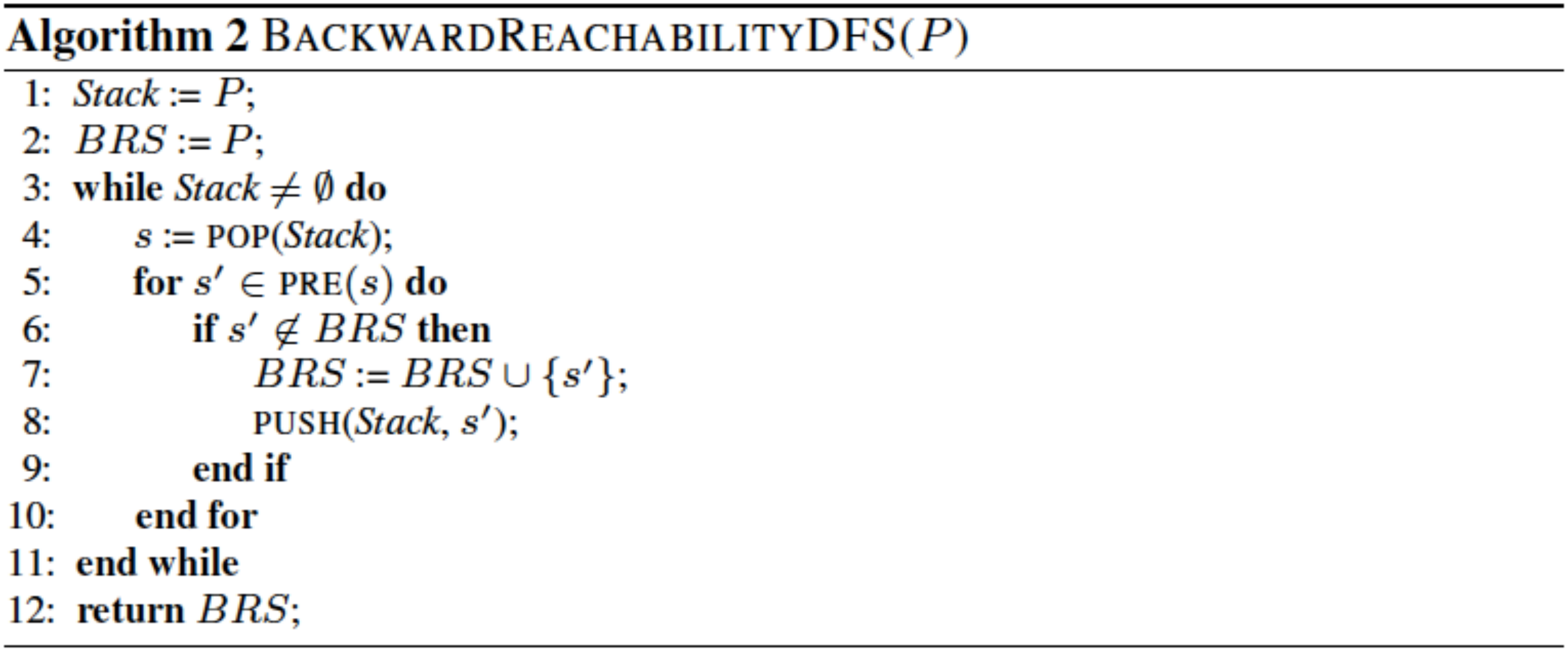
Algorithm 1 REACHABILITYDFS

```
$Stack := I$;
$RS := I$;
while $Stack \neq \emptyset$ do
    $s :=$ POP($Stack$);
    $s' :=$ POST($s$);
    if $s' \notin RS$ then
        $RS := RS \cup \{s'\}$;
        PUSH($Stack$, $s'$);
    end if
end while
return $RS$;
```

Pre-condition function

$$
\begin{aligned}
s_2 \in \textsc{pre}(s_1, l) &\Leftrightarrow (s_2, s_1) \in r_l \\
s_2 \in \textsc{pre}(s_1) &\Leftrightarrow \exists l \in L \ : \ s_2 \in \textsc{pre}(s_1, r_l) \\
s_2 \in \textsc{pre}(s_1) &\Leftrightarrow (s_2, s_1) \in R
\end{aligned}
$$

Backward reachability using DFS



Algorithm 2 BackwardReachabilityDFS(P)

```
Stack := $P$;
$BRS := P$;
while $Stack \neq \emptyset$ do
    $s :=$ POP($Stack$);
    for $s' \in$ PRE$(s)$ do
        if $s' \notin BRS$ then
            $BRS := BRS \cup \{s'\}$;
            PUSH($Stack$, $s'$);
        end if
    end for
end while
return $BRS$;
```

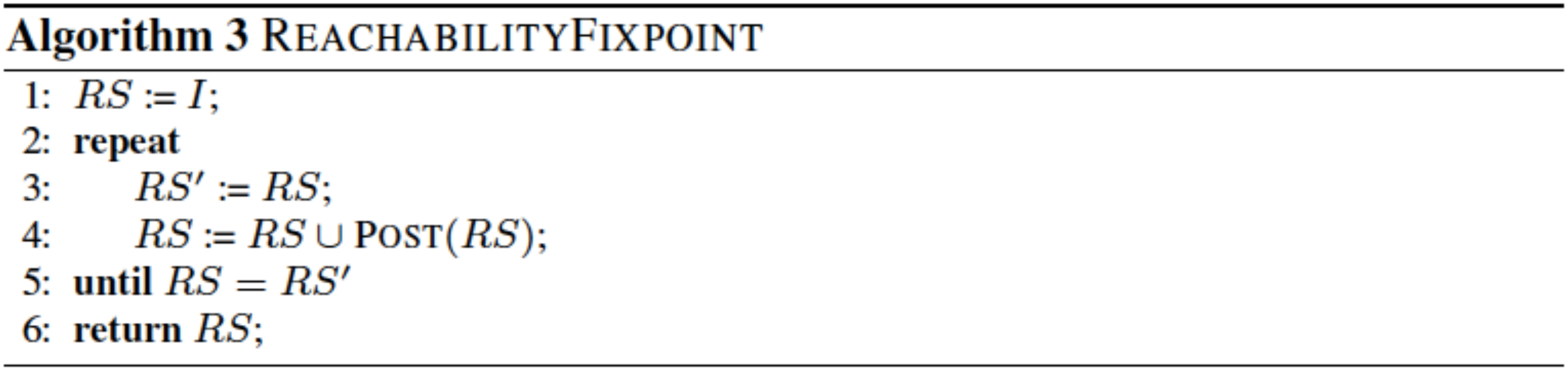
Explicit vs. Symbolic reachability analysis

- The DFS algorithms that we showed work on one state at a time. This is called explicit state (or enumerative, or concrete) reachability analysis
- It is not feasible to enumerate each state since state space of a program is exponential in the number of variables
- Symbolic reachability analysis works on sets of states, rather than a single state at a time
- We need to generalize pre and post condition functions so that they work on sets of states

Post and pre condition

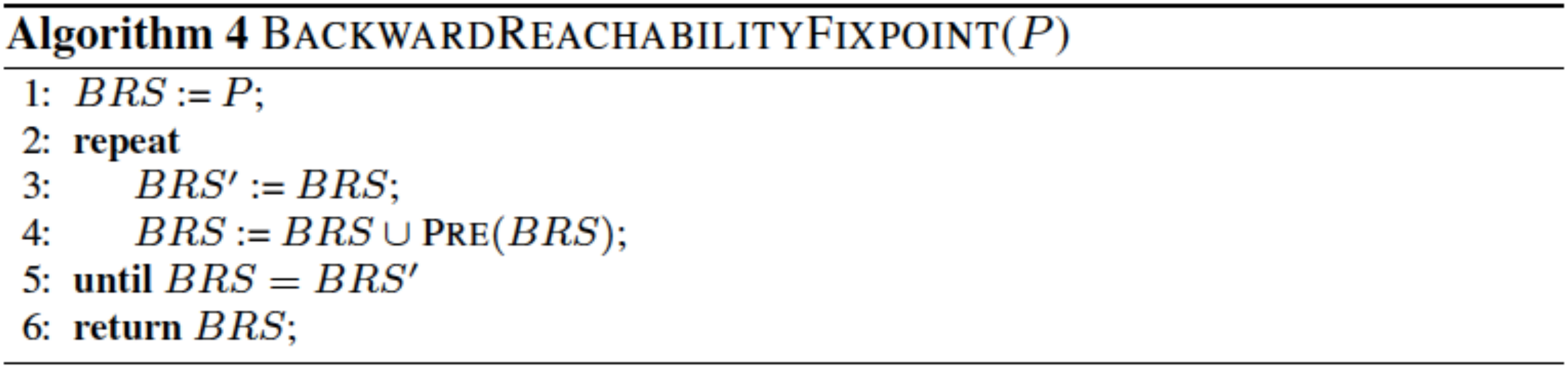
$$\begin{aligned}
\textsc{Post}(P, l) &= \{s \mid \exists s' \in P \ : \ (s', s) \in r_l\} \\
\textsc{Post}(P) &= \{s \mid \exists s' \in P \ : \ (s', s) \in R\} \\
\textsc{Pre}(P, l) &= \{s \mid \exists s' \in P \ : \ (s, s') \in r_l\} \\
\textsc{Pre}(P) &= \{s \mid \exists s' \in P \ : \ (s, s') \in R\}
\end{aligned}$$

Symbolic Reachability Analysis



Algorithm 3 ReachabilityFixpoint

```
$RS := I$;
repeat
    $RS' := RS$;
    $RS := RS \cup \textsc{Post}(RS)$;
until $RS = RS'$
return $RS$;
```



Algorithm 4 BackwardReachabilityFixpoint(P)

```
$BRS := P$;
repeat
    $BRS' := BRS$;
    $BRS := BRS \cup \textsc{Pre}(BRS)$;
until $BRS = BRS'$
return $BRS$;
```

Reachability and fixpoints

- We will demonstrate that reachability analysis corresponds to computing the least fixpoint of a function.

- In order to do that we need to introduce the concept of a lattice

Pre and post condition functions on sets of states

- Given a transition system T=(S, I, R), we define functions from sets of states to sets of states
  - $\mathcal{F} : 2^S \to 2^S$
- For example, one such function is the post function (which computes the post-condition of a set of states)
  - post : $2^S \to 2^S$

  which can be defined as (where $P \subseteq S$):

  $\text{Post}(P) = \{ s' \mid (s,s') \in R \text{ and } s \in P \}$
- We can similarly define the pre function (which computes the pre-condition of a set of states)
  - pre : $2^S \to 2^S$

  which can be defined as:

  $\text{Pre}(P) = \{ s \mid (s,s') \in R \text{ and } s' \in P \}$

Lattices

The set of states of the transition system forms a lattice:

lattice	2^S
partial order	$\subseteq$
bottom element	$\varnothing$ (alternative notation: $\perp$)
top element	S (alternative notation: T)
Least upper bound (lub) (aka join) operator	$\cup$
Greatest lower bound (glb) (aka meet) operator	$\cap$

Lattices

In general, a lattice is a partially ordered set with a least upper bound operation and a greatest lower bound operation.

- Least upper bound $a \cup b$ is the smallest element where

  $a \subseteq a \cup b$ and $b \subseteq a \cup b$
- Greatest lower bound $a \cap b$ is the biggest element where

  $a \cap b \subseteq a$ and $a \cap b \subseteq b$

A partial order is a

- reflexive (for all x, $x \subseteq x$),
- transitive (for all x, y, z, $x \subseteq y \wedge y \subseteq z \Rightarrow x \subseteq z$), and
- antisymmetric (for all x, y, $x \subseteq y \wedge y \subseteq x \Rightarrow x = y$)

relation.

Complete Lattices

2^S forms a lattice with the partial order defined as the subset-or-equal relation and the least upper bound operation defined as the set union and the greatest lower bound operation defined as the set intersection.

In fact, $(2^S, \subseteq, \emptyset, S, \cup, \cap)$ is a complete lattice since for each set of elements from this lattice there is a least upper bound and a greatest lower bound.

Also, note that the top and bottom elements can be defined as:

$\bot = \emptyset = \cap \{ y \mid y \in 2^S \}$

$T = S = \cup \{ y \mid y \in 2^S \}$

This definition is valid for any complete lattice.

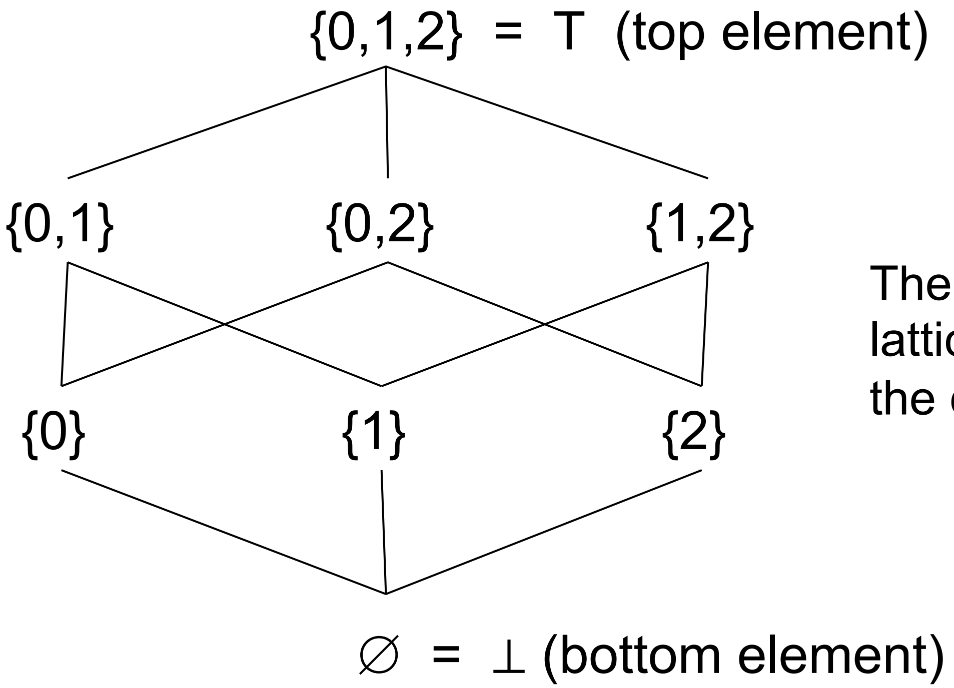
An Example Lattice

{∅, {0}, {1}, {2}, {0,1},{0,2},{1,2},{0,1,2}}

partial order: ⊆ (subset relation)

bottom element: ∅ = ⊥ top element: {0,1,2} = T

lub: ∪ (union) glb: ∩ (intersection)



The Hasse diagram for the example lattice (shows the transitive reduction of the corresponding partial order relation)

What is a Fixpoint (aka, Fixed Point)

Given a function

$\mathcal{F} : D \rightarrow D$

$x \in D$ is a fixpoint of $\mathcal{F}$ if and only if $\mathcal{F}(x) = x$

Reachability

Let RS(I) denote the set of states reachable from the initial states I of the transition system T = (S, I, R)

In general, given a set of states $P \subseteq S$, we can define the reachability function as follows:

$RS(P) = \{s_n \mid s_n \in P$, or there exists $s_0 s_1 \ldots s_n \in S$,
where for all $0 \leq i < n$ $(s_i, s_{i+1}) \in R$, and $s_0 \in P \}$

We can also define the backward reachability function BRS as follows:

$BRS(P) = \{s_0 \mid s_0 \in P$, or there exists $s_0 s_1 \ldots s_n \in S$,
where for all $0 \leq i < n$ $(s_i, s_{i+1}) \in R$, and $s_n \in P \}$

Reachability ≡ Fixpoints

Here is an interesting property

$$RS(P) = P \cup post(RS(P))$$

we observe that RS(P) is a fixpoint of the following function:

$\mathcal{F}\, y = P \cup post(y)$ (we can also write it as $\lambda\, y\, .\, P \cup post(y)$)

$\mathcal{F}\,(RS(P)) = RS(P)$

In fact, RS(P) is the least fixpoint of $\mathcal{F}$, which is written as:

$$RS(P) = \mu\, y\, .\, \mathcal{F}\, y = \mu\, y\, .\, P \cup post(y)$$

(μ means least fixpoint)

Reachability ≡ Fixpoints

We have the same property for backward reachability

$$\mathrm{BRS}(P) = P \cup \mathrm{pre}(\mathrm{RS}(P))$$

i.e., BRS(P) is a fixpoint of the following function:

$\mathcal{F}\, y = P \cup \mathrm{pre}(y)$ (we can also write it as $\lambda\, y \,.\, P \cup \mathrm{pre}(y)$)

$$\mathcal{F}(\mathrm{RS}(P)) = \mathrm{RS}(P)$$

In fact, BRS(P) is the least fixpoint of $\mathcal{F}$, which is written as:

$$\mathrm{BRS}(P) = \mu\, y \,.\, \mathcal{F}\, y = \mu\, y \,.\, P \cup \mathrm{pre}(y)$$

$RS(P) = \mu\, y\,.\, P \cup RS(y)$

- Let’s prove this.

- First we have the equivalence $RS(P) = P \cup post(RS(P))$
  - Why? Because according to the definition of RS(P), a state is in RS(P) if that state is in P, or if that state has a previous state which is in RS(P).
  - From this equivalence we know that RS(P) is a fixpoint of the function $\lambda\, y\,.\, P \cup post(y)$ and since the least fixpoint is the smallest fixpoint we have:

    $\mu\, y\,.\, P \cup post(y) \subseteq RS(P)$

$RS(P) = \mu y . P \cup RS(y)$

- Next we need to prove that $RS(P) \subseteq \mu y . P \cup RS(y)$ to complete the proof.
- Suppose z is a fixpoint of $\lambda y . P \cup RS(y)$, then we know that $z = P \cup RS(z)$ which means that $RS(z) \subseteq z$ and this means that no state that is reachable from z is outside of z.
- Since we also have $P \subseteq z$, any path that is reachable from P must be in z.

Hence, we can conclude that $RS(P) \subseteq z$.

Since we showed that RS(P) is contained in any fixpoint of the function $\lambda y . P \cup RS(y)$, we get

$RS(P) \subseteq \mu y . P \cup RS(y)$

which completes the proof.

Monotonicity

- Function $\mathcal{F}$ is monotonic if and only if, for any x and y,
  $x \subseteq y \Rightarrow \mathcal{F}\, x \subseteq \mathcal{F}\, y$

Note that,

$\lambda\, y\, .\, P \cup post(y)$

$\lambda\, y\, .\, P \cup pre(y)$

are monotonic.

For both these functions, if you give a bigger y as input you will get a bigger result as output.

Monotonicity

- One can define non-monotonic functions:

For example: $\lambda\ y\ .\ P\ \cup \mathrm{post}(S - y)$

This function is not monotonic. If you give a bigger y as input you will get a smaller result.

- For the functions that are non-monotonic the fixpoint computation techniques we are going to discuss will not work. For such functions a fixpoint may not even exist.
- The functions we defined for reachability are monotonic because we are applying monotonic operations (like post and $\cup$) to the input variable y.
- Set complement – is not monotonic. However, if you have an even number of negations in front of the input variable y, then you will get a monotonic function.

Least Fixpoint

Given a monotonic function $\mathcal{F}$, its least fixpoint exists, and it is the greatest lower bound (glb) of all the reductive elements :

$$\mu\, y\, .\, \mathcal{F}\, y = \cap \{\, y \mid \mathcal{F}\, y \subseteq y \,\}$$

$\mu\, y\,.\, \mathcal{F}\, y = \cap\,\{\, y \mid \mathcal{F}\, y \subseteq y\,\}$

- Let's prove this property.
- Let us define z as $z = \cap\,\{\, y \mid \mathcal{F}\, y \subseteq y\,\}$

We will first show that z is a fixpoint of $\mathcal{F}$ and then we will show that it is the least fixpoint which will complete the proof.

- Based on the definition of z, we know that:

  for any y, $\mathcal{F}\, y \subseteq y$, we have $z \subseteq y$.

Since $\mathcal{F}$ is monotonic, $z \subseteq y \Rightarrow \mathcal{F}\, z \subseteq \mathcal{F}\, y$.

But since $\mathcal{F}\, y \subseteq y$, then $\mathcal{F}\, z \subseteq y$.

I.e., for all y, $\mathcal{F}\, y \subseteq y$, we have $\mathcal{F}\, z \subseteq y$.

This implies that, $\mathcal{F}\, z \subseteq \cap\,\{\, y \mid \mathcal{F}\, y \subseteq y\,\}$,

and based on the definition of z, we get $\mathcal{F}\, z \subseteq z$

$\mu\, y \,.\, \mathcal{F}\, y = \cap \{ y \mid \mathcal{F}\, y \subseteq y \}$

- Since $\mathcal{F}$ is monotonic and since $\mathcal{F}\, z \subseteq z$, we have $\mathcal{F}\, (\mathcal{F}\, z) \subseteq \mathcal{F}\, z$ which means that $\mathcal{F}\, z \in \{ y \mid \mathcal{F}\, y \subseteq y \}$. Then by definition of z we get, $z \subseteq \mathcal{F}\, z$

- Since we showed that $\mathcal{F}\, z \subseteq z$ and $z \subseteq \mathcal{F}\, z$, we conclude that $\mathcal{F}\, z = z$, i.e., z is a fixpoint of the function $\mathcal{F}$.

- For any fixpoint of $\mathcal{F}$ we have $\mathcal{F}\, y = y$ which implies $\mathcal{F}\, y \subseteq y$ So any fixpoint of $\mathcal{F}$ is a member of the set $\{ y \mid \mathcal{F}\, y \subseteq y \}$ and z is smaller than any member of the set $\{ y \mid \mathcal{F}\, y \subseteq y \}$ since it is the greatest lower bound of all the elements in that set.

Hence, z is the least fixpoint of $\mathcal{F}$.

Computing the Least Fixpoint

The least fixpoint $\mu\, y\, .\, \mathcal{F}\, y$ is the limit of the following sequence (assuming $\mathcal{F}$ is $\cup$-continuous):

$\emptyset, \mathcal{F}\,\emptyset, \mathcal{F}^2\,\emptyset, \mathcal{F}^3\,\emptyset, \ldots$

$\mathcal{F}$ is $\cup$-continuous if and only if

$p_1 \subseteq p_2 \subseteq p_3 \subseteq \ldots$ implies that $\mathcal{F}\,(\cup_i\, p_i) = \cup_i\, \mathcal{F}\,(p_i)$

If S is finite, then we can compute the least fixpoint using the sequence $\emptyset, \mathcal{F}\,\emptyset, \mathcal{F}^2\,\emptyset, \mathcal{F}^3\,\emptyset, \ldots$ This sequence is guaranteed to converge if S is finite and it will converge to the least fixpoint.

Computing the Least Fixpoint

Given a monotonic and union continuous function $\mathcal{F}$

$$\mu\, y\, .\, \mathcal{F}\, y = \cup_i\, \mathcal{F}^i\, (\emptyset)$$

We can prove this as follows:

- First, we can show that for all i, $\mathcal{F}^i\, (\emptyset) \subseteq \mu\, y\, .\, \mathcal{F}\, y$ using induction

for i=0, we have $\mathcal{F}^0\, (\emptyset) = \emptyset \subseteq \mu\, y\, .\, \mathcal{F}\, y$

Assuming $\mathcal{F}^i\, (\emptyset) \subseteq \mu\, y\, .\, \mathcal{F}\, y$

and applying the function $\mathcal{F}$ to both sides and using monotonicity of $\mathcal{F}$ we get: $\mathcal{F}\, (\mathcal{F}^i\, (\emptyset)) \subseteq \mathcal{F}\, (\mu\, y\, .\, \mathcal{F}\, y)$

and since $\mu\, y\, .\, \mathcal{F}\, y$ is a fixpoint of $\mathcal{F}$ we get:

$\mathcal{F}^{i+1}\, (\emptyset) \subseteq \mu\, y\, .\, \mathcal{F}\, y$

which completes the induction.

Computing the Least Fixpoint

- So, we showed that for all i, $\mathcal{F}^{i}(\emptyset) \subseteq \mu y\,.\,\mathcal{F}\,y$
- If we take the least upper bound of all the elements in the sequence $\mathcal{F}^{i}(\emptyset)$ we get $\cup_i \mathcal{F}^{i}(\emptyset)$ and using above result, we have:

$$\cup_i \mathcal{F}^{i}(\emptyset) \subseteq \mu y\,.\,\mathcal{F}\,y$$

- Now, using union-continuity we can conclude that

$$\mathcal{F}(\cup_i \mathcal{F}^{i}(\emptyset)) = \cup_i \mathcal{F}(\mathcal{F}^{i}(\emptyset)) = \cup_i \mathcal{F}^{i+1}(\emptyset)$$
$$= \emptyset \cup_i \mathcal{F}^{i+1}(\emptyset) = \cup_i \mathcal{F}^{i}(\emptyset)$$

- So, we showed that $\cup_i \mathcal{F}^{i}(\emptyset)$ is a fixpoint of $\mathcal{F}$ and $\cup_i \mathcal{F}^{i}(\emptyset) \subseteq \mu y\,.\,\mathcal{F}\,y$, then we conclude that $\mu y\,.\,\mathcal{F}\,y = \cup_i \mathcal{F}^{i}(\emptyset)$

Computing the Least Fixpoint

If there exists a j, where $\mathcal{F}^{j}(\emptyset) = \mathcal{F}^{j+1}(\emptyset)$, then

$$\mu y \,.\, \mathcal{F}\, y = \mathcal{F}^{j}(\emptyset)$$

- We have proved earlier that for all i, $\mathcal{F}^{i}(\emptyset) \subseteq \mu y \,.\, \mathcal{F}\, y$

- If $\mathcal{F}^{j}(\emptyset) = \mathcal{F}^{j+1}(\emptyset)$, then $\mathcal{F}^{j}(\emptyset)$ is a fixpoint of $\mathcal{F}$ and since we know that $\mathcal{F}^{j}(\emptyset) \subseteq \mu y \,.\, \mathcal{F}\, y$ then we conclude that

$$\mu y \,.\, \mathcal{F}\, y = \mathcal{F}^{j}(\emptyset)$$

RS(P) Fixpoint Computation

$RS(P) = \mu\, y \,.\, P \cup RS(y)$ is the limit of the sequence:

$\emptyset$,

$P \cup post(\emptyset)$,

$P \cup post(P \cup post(\emptyset))$,

$P \cup post(P \cup post\ (p \cup post(\emptyset)))$

, ...

which is equivalent to

$\emptyset$, P, $P \cup post(P)$, $P \cup post(P \cup post(P)\ )$, ...

RS(P) Fixpoint Computation

RS(P) ≡ states that are reachable from P ≡ $P \cup \mathrm{post}(P) \cup \mathrm{post}(\mathrm{post}(P)) \cup \ldots$