String Constraints for Verification (CAV'14)

Parosh Aziz Abdulla[1] Mohamed Faouzi Atig[1] Yu-Fang Chen[2]
Lukáš Holík[3] Ahmed Rezine[4] Philipp Rümmer[1] Jari Stenman[1]

Department of Information Technology, Uppsala University, Sweden

Institute of Information Science, Academia Sinica, Taiwan

Faculty of Information Technology, Brno University of Technology, Czech Republic

Department of Computer and Information Science, Linköping University, Sweden

iPRA 2014, July 18, 2014

Table of Contents

Table of Contents

Example Program

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ϵ)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

Example Program

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ε)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

$s \neq s_1 \cdot ba \cdot s_2$

Example Program

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ϵ)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

$s = u \cdot v$

Example Program

$$s \in \epsilon$$

$$u \in a^* \wedge v \in b^*$$

```
// Pre = (true)
String s='';
// P_1 = (s ∈ ϵ)
while(*){
    // P_2 = (s = u · v ∧ u ∈ a^* ∧ v ∈ b^* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P_3 = P_2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P_3
```

$$s \notin \Sigma^* \cdot ba \cdot \Sigma^*$$

Example Program

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ε)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

$|u| = |v|$

$|s| = 2n$

Table of Contents

Procedure Overview

Problem

Given: Set of constraints containing:

1. Disequalities $XY \neq aZb$
2. Equalities $XY = aZb$
3. Memberships $aXYb \in (abb)^*$
4. Length constraints $|X| = |Y| + |Z|$

Task: Report Sat, together with a satisfying assignment for variables (X, Y, Z), or Unsat.

Procedure Overview

Problem

Given: Set of constraints containing:

1. Disequalities $XY \neq aZb$
2. Equalities $XY = aZb$
3. Memberships $aXYb \in (abb)^*$
4. Length constraints $|X| = |Y| + |Z|$

Task: Report Sat, together with a satisfying assignment for variables (X, Y, Z), or Unsat.

Disequalities

Example

- Assume $\Sigma = \{a, b\}$

$$X, Y \in (ab)^* \wedge X \neq Y$$

Disequalities

Example

- Assume $\Sigma = \{a, b\}$

$$X, Y \in (ab)^* \wedge X \neq Y$$

$$X, Y \in (ab)^* \wedge$$
$$X = UaV \wedge Y = UbV'$$

Disequalities

Example

- Assume $\Sigma = \{a, b\}$

$$X, Y \in (ab)^* \wedge X \neq Y$$

$$X, Y \in (ab)^* \wedge \quad X = UaV \wedge Y = UbV'$$

$$X, Y \in (ab)^* \wedge \quad X = UbV \wedge Y = UaV'$$

Disequalities

Example

- Assume $\Sigma = \{a, b\}$

$$X, Y \in (ab)^* \wedge X \neq Y$$

$$X, Y \in (ab)^* \wedge \quad X = UaV \wedge Y = UbV'$$

$$X, Y \in (ab)^* \wedge \quad X = UbV \wedge Y = UaV'$$

$$X, Y \in (ab)^* \wedge \quad |X| \neq |Y|$$

Procedure Overview

Problem

Given: Set of constraints containing:

1. ~~Disequalities $XY \neq aZb$~~
2. Equalities $XY = aZb$
3. Memberships $aXYb \in (abb)^*$
4. Length constraints $|X| = |Y| + |Z|$

Task: Report Sat, together with a satisfying assignment for variables (X, Y, Z), or Unsat.

Equalities

Example

$$Y \in (ab)^* \wedge XY = bZb$$

Equalities

Example

$$Y \in (ab)^* \wedge XY = bZb$$

$$Y \in (ab)^* \wedge X = \epsilon \wedge Y = bZb$$

Equalities

Example

$$Y \in (ab)^* \wedge XY = bZb$$

$$Y \in (ab)^* \wedge X = \epsilon \wedge Y = bZb$$

$$bZb \in (ab)^*$$

Equalities

Example

$$Y \in (ab)^* \wedge XY = bZb$$

$$Y \in (ab)^* \wedge X = \epsilon \wedge Y = bZb \qquad Y \in (ab)^* \wedge X = bZ_1 \wedge Y = Z_2 b \wedge Z = Z_1 Z_2$$

$$bZb \in (ab)^*$$

Equalities

Example

$$Y \in (ab)^* \wedge XY = bZb$$

$$Y \in (ab)^* \wedge X = \epsilon \wedge Y = bZb \qquad Y \in (ab)^* \wedge X = bZ_1 \wedge Y = Z_2 b \wedge Z = Z_1 Z_2$$

$$bZb \in (ab)^* \qquad Z_2 b \in (ab)^*$$

Procedure Overview

Problem

Given: Set of constraints containing:

1. ~~Disequalities $XY \neq aZb$~~
2. ~~Equalities $XY \neq aZb$~~
3. Memberships $aXYb \in (abb)^*$
4. Length constraints $|X| = |Y| + |Z|$

Task: Report Sat, together with a satisfying assignment for variables (X, Y, Z), or Unsat.

Memberships

General Memberships

1. Reduce to **simple** memberships ($X \in R$).
2. Solve simple memberships.

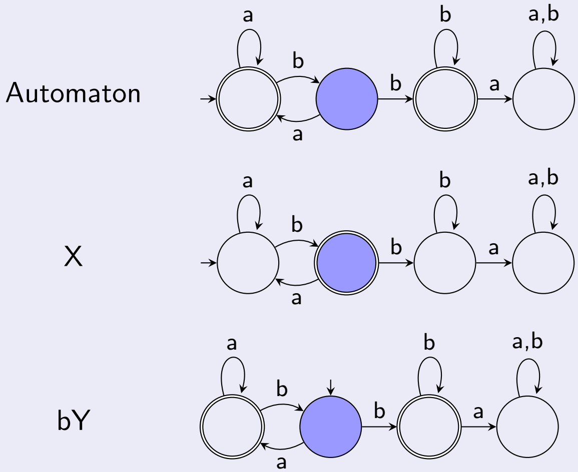
Memberships

General Memberships

1. Reduce to **simple** memberships ($X \in R$).
2. Solve simple memberships.

Memberships

$XbY \in a^*(ba)^*bb^*a^*(a|b)^*$



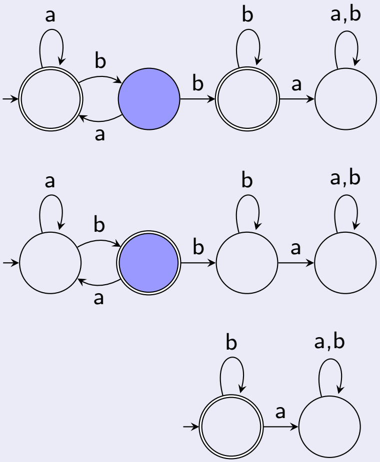
Memberships

$XbY \in a^*(ba)^*bb^*a^*(a|b)^*$

Automaton

X

Y



Memberships

General Memberships

1. Reduce to **simple** memberships ($X \in R$).
2. Solve simple memberships.

Example

$$|X| \leq 10 \wedge X \in (aaa)^* \wedge X \in (aaaa)^*$$

Memberships

General Memberships

1. Reduce to **simple** memberships ($X \in R$).
2. Solve simple memberships.

Example

$$|X| \leq 10 \land X \in (aaa)^* \land X \in (aaaa)^*$$

$$|X| \leq 10 \land X \in (aaa)^* \cap (aaaa)^*$$

Memberships

General Memberships

1. Reduce to **simple** memberships ($X \in R$).
2. Solve simple memberships.

Example

$$|X| \leq 10 \wedge X \in (aaa)^* \wedge X \in (aaaa)^*$$

$$|X| \leq 10 \wedge X \in (aaa)^* \cap (aaaa)^*$$

$$|X| \leq 10 \wedge |X| = 12n$$

Procedure Overview

Problem

Given: formula containing:

1. ~~Disequalities $XY \neq aZb$~~
2. ~~Equalities $XY = aZb$~~
3. ~~Memberships $aXYb \in (abb)^*$~~
4. Length constraints $|X| = |Y| + |Z|$

Task: find satisfying assignment for variables (X, Y, Z).

Procedure Overview

Problem

Given: formula containing:

1. ~~Disequalities $XY \neq aZb$~~
2. ~~Equalities $XY = aZb$~~
3. ~~Memberships $aXYb \in (abb)^*$~~
4. Length constraints $|X| = |Y| + |Z|$

Task: find satisfying assignment for variables (X, Y, Z).

Length Constraints

Solve using some existing decision procedure.

Soundness and Completeness

Soundness

All inference rules are sound.

Completeness

In general, splitting of equalities might not terminate.

- Graph-based **acyclicity** condition to detect this case.
- Procedure terminates on acyclic formulae.
- All constraints we've encountered in practice have been acyclic.

Table of Contents

Verification with Horn Clauses

- Horn clauses as an intermediate representation:
  Clauses represent **Programs + Verification methodology**
  - Floyd-Hoare proofs
  - Design/verification by contract
  - Owicki-Gries
  - Rely-Guarantee
- Horn clauses are constructed such that:
  Clauses are solvable iff Program is correct
- Separation of concerns:
  1. Representation of problem with Horn clauses
  2. Of-the-shelf solver for Horn clauses
- Elegant way to generalise existing algorithms to inter-procedural or concurrent analysis
  (e.g., predicate abstraction, abstract interpretation)

[Grebenshchikov, Lopes, Popeea, Rybalchenko, PLDI'12]

Verification with Horn Clauses

Program

```
String s= '';
while(*){
    s= 'a' + s + 'b';
}
assert(!s.contains('ba') && (s.length() % 2) == 0);
```

Horn Clauses

$$P_0(s) \leftarrow true$$

$$P_0(a \cdot s \cdot b) \leftarrow P_0(s)$$

$$P_1(s) \leftarrow P_0(s)$$

$$false \leftarrow P_1(s) \wedge s \in \Sigma^* \cdot ba \cdot \Sigma^*$$

$$\exists n.|s| = 2n \leftarrow P_1(s)$$

Verification with Horn Clauses

CEGAR Loop

1. Fix set of predicates for each relation symbol
2. Build abstract reachability graph
3. If **false** is reachable, extract counterexample
4. Check counterexample:
   - **Genuine**: Return counterexample
   - **Spurious**: Generate new predicates

Verification with Horn Clauses

CEGAR Loop

1. Fix set of predicates for each relation symbol
2. Build abstract reachability graph
3. If **false** is reachable, extract counterexample
4. Check counterexample:
   - **Genuine**: Return counterexample
   - **Spurious**: Generate new predicates

Predicate Synthesis

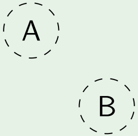
- Interpolants are good predicates
- 2-layered interpolation approach:
  - Try to synthesize length interpolant
  - Try to synthesize regular interpolant $s_1|s_n|\cdots|s_n \in \mathcal{R}$

Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset$; $Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example

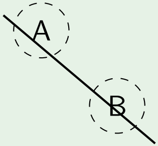


Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset$; $Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example



Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset; \quad Bw \leftarrow \emptyset$

while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**

  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**

    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$

  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**

    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$

  else

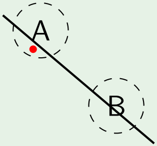
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$

  end if

end while

return Inseparable

Example

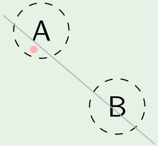


Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset; \quad Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example

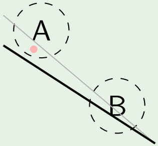


Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset; \quad Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example

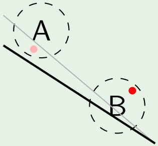


Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset$; $Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example

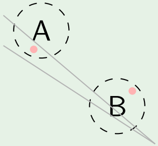


Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset; \quad Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example



Regular Interpolation

Algorithm

$Aw \leftarrow \emptyset$; $Bw \leftarrow \emptyset$
while there is RE $\mathcal{R}$ of size $\leq L$ such that $Aw \subseteq \mathcal{L}(\mathcal{R})$ and $Bw \cap \mathcal{L}(\mathcal{R}) = \emptyset$ **do**
  if $A[\bar{s}] \wedge \neg(s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Aw \leftarrow Aw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else if $B[\bar{s}] \wedge (s_1|s_2|\cdots|s_n \in \mathcal{R})$ sat with assignment η **then**
    $Bw \leftarrow Bw \cup \{\eta(s_1)|\cdots|\eta(s_n)\}$
  else
   return $s_1|s_2|\cdots|s_n \in \mathcal{R}$
  end if
end while
return Inseparable

Example

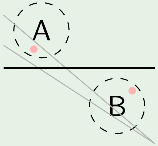


Table of Contents

Implementation

Norn

- Implemented in Scala
- Uses Princess for Linear Arithmetic

Model Checker

- Based on Horn clauses and interpolation.
- Uses Norn as a backend.

Program	**# of Calls to solver**	**Time**
$a^n b^n$	168	8s
StringReplace	59	4.5s
ChunkSplit	58	5.5s
Levenshtein	87	5.3s
HammingDistance	1493	27.1s

Conclusions

Verification of String Programs

- Procedure for checking satisfiability of string formulae
- New: unbounded variables together with language constraints
- Complete for expressive fragment of the logic
- Horn clause-based Model Checker

Future Work

Theory

- Expressiveness
- Interpolation

Tool

- Performance
- DPLL(T)
- Applications

Thank you!