

## From Büchi automata to S1S

Given automaton  $(\{a_1, \dots, a_n\}, \{q_1, \dots, q_m\}, S, F, \delta)$ .

Consider variables  $X_i$  for  $1 \leq i \leq n$  and  $Y_i$  for  $1 \leq i \leq m$ .

$X_i$  denotes the positions in the input where  $a_i$  occurs.

$Y_i$  denotes positions in the run where  $q_i$  occurs.

The required formula  $\phi(X_1, \dots, X_n)$  is

$\exists Y_1 \dots \exists Y_m$ .

$\text{partition}(X_1, \dots, X_n) \wedge \text{partition}(Y_1, \dots, Y_m)$

$\wedge \bigvee_{q_i \in S} O \in Y_i$

$\wedge \forall x (\bigvee_{q_k \in \delta(q_i, a_j)} x \in Y_i \wedge x \in X_j \wedge S(x) \in X_k)$

$\wedge \bigvee_{q_i \in S} \text{infinite}(Y_i)$ .

$$\textit{partition}(Z_1, \dots, Z_k) \equiv \forall x (\bigvee_{1 \leq i \leq k} (x \in Z_i) \wedge \bigwedge_{1 \leq i \leq k} (x \in Z_i \Rightarrow \bigwedge_{j \neq i} x \notin Z_j)).$$

$$\textit{infinite}(X) \equiv \forall x \exists y (x < y \wedge y \in X).$$

Hence all languages accepted by Büchi automata can be defined in S1S.

**WS1S** : weak monadic second order logic of one successor.

Same syntax as for S1S.

Same semantics, except that second order variables are interpreted as **finite** subsets of  $\mathbb{N}$ .

WS1S has the same expressive power as automata on finite words.

**WSkS** : weak monadic second order logic of  $k$  successors.

Terms now denote positions in a tree instead of in a linear sequence.

Terms are of the form

$\epsilon$  (the empty string)

$x, y, \dots$  (first order variables)

$t_1, \dots, t_k$

Atomic formulas:  $t_1 = t_2, t \in X$  where  $X$  is a second order variable.

Logical connectives and quantifiers as usual.

Semantics: terms represent strings from  $\{1, \dots, k\}^*$ . Second order variables are interpreted as finite sets of such strings.

## Example formulas:

–  $X \subseteq Y \equiv \forall x(x \in X \Rightarrow x \in Y)$

– Union:  $X = X_1 \cup X_2 \equiv X_1 \subseteq X \wedge X_2 \subseteq X$   
 $\wedge \forall x(x \in X \Rightarrow (x \in X_1 \vee x \in X_2))$

– Intersection:

$$X = X_1 \cap X_2 \equiv X \subseteq X_1 \wedge X \subseteq X_2 \wedge \forall x(x \in X_1 \wedge x \in X_2 \Rightarrow x \in X)$$

– Emptiness:  $X = \emptyset \equiv \forall Y(Y \subseteq X \Rightarrow X \subseteq Y)$

– *Partition*( $X, X_1, \dots, X_n$ )  $\equiv$

$$X = X_1 \cup \dots \cup X_n \wedge \bigwedge_{i \neq j} X_i \cap X_j = \emptyset$$

– The prefix ordering:

$$x \leq y \equiv \forall X(y \in X \wedge (\forall z(\bigvee_{i=1}^k (z_i \in X)) \Rightarrow z \in X) \Rightarrow x \in X)$$

– *PrefixClosed*( $X$ )  $\equiv \forall x \forall y(x \in X \wedge y \leq x \Rightarrow y \in X)$

## Encoding of trees as sets:

Let  $t \in T(\Sigma)$ , and the maximal arity of the symbols in  $\Sigma = \{f_1, \dots, f_n\}$  be  $k$ .

We encode  $t$  as the tuple of sets  $C(t) = (S, S_{f_1}, \dots, S_{f_n})$  where  $S_{f_i}$  is the set of positions in  $t$  which are labeled with  $f_i$  and  $S = S_{f_1} \cup \dots \cup S_{f_n}$ .

The term  $f(g(a), f(a, b))$  is encoded as  $S = \{\epsilon, 1, 11, 2, 21, 22\}$ ,  
 $S_f = \{\epsilon, 2\}$ ,  $S_g = \{1\}$ ,  $S_a = \{11, 21\}$ ,  $S_b = \{22\}$

$$\begin{aligned} \text{Term}(X, X_{f_1}, \dots, X_{f_n}) &\equiv X \neq \emptyset \wedge \text{PrefixClosed}(X) \\ &\wedge \text{Partition}(X, X_{f_1}, \dots, X_{f_n}) \\ &\wedge \bigwedge_{i=1}^k \bigwedge_{f_j} \text{has arity } i \left( \bigwedge_{l=1}^i \forall x (x \in X_{f_j} \Rightarrow xl \in X) \right. \\ &\quad \left. \wedge \bigwedge_{l=i+1}^k \forall x (x \in X_{f_j} \Rightarrow xl \notin X) \right) \end{aligned}$$

Conversely given sets  $S_1, \dots, S_n \subseteq \{1, \dots, k\}^*$  we interpret it as a tree.

The positions of this tree are

$$\{\epsilon\} \cup \{pi \mid \exists p' \in \bigcup_{i=1}^n S_i, p \leq p', i \in \{1, \dots, k\}\}.$$

The symbol at position  $p$  is  $(a_1, \dots, a_n)$  where

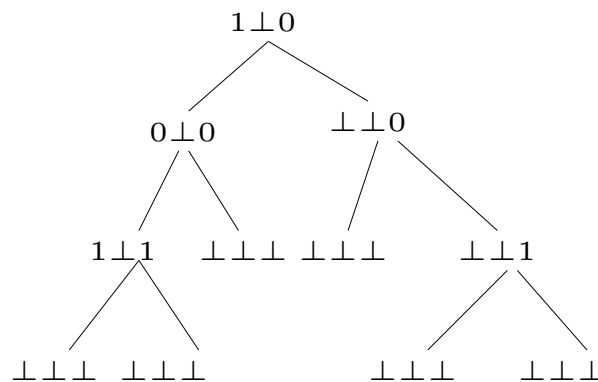
$$a_i = 1 \text{ iff } p \in S_i$$

$$a_i = 0 \text{ iff } p \notin S_i \text{ and } p \leq q \text{ for some } q \in S_i$$

$$a_i = \perp \text{ otherwise}$$

$(a_1, \dots, a_n)$  is a constant if each  $a_i = \perp$ , otherwise it has arity  $k$ .

E.g. the tuple  $(\{\epsilon, 11\}, \emptyset, \{11, 22\})$  represents the tree



## From WSkS to tree automata

First perform some simplifications:

$ti = ui$  simplifies to  $t = u$

$ti = uj$  simplifies to  $\perp$  for  $i \neq j$

$ti = \epsilon$  simplifies to  $\perp$

$x = ti$  simplifies to  $\exists y(x = yi \wedge y = t)$  (where  $t$  is not a variable and  $y$  is fresh)

$t \in X$  simplifies to  $\exists y(t = y \wedge y \in X)$

After repeated applications all atomic formulas are of the form  $x = \epsilon$ ,  $x = yi$  and  $x \in X$ .

We introduce following atomic formulas:

- $X \subseteq Y$
- $X = Yi$  (interpreted as:  $X$  and  $Y$  are singletons  $\{x\}$  and  $\{y\}$ , and  $x = yi$ )
- $X = \epsilon$

As in the case of S1S we eliminate all first order variables, using the formula *Singleton*.

Finally all atomic formulas are of the form  $X \subseteq Y$ ,  $X = Yi$  and  $X = \epsilon$ .

Automaton for  $X \subseteq Y$ , with final state  $q$ :

$$\begin{array}{lll} \perp\perp \rightarrow q & 00(q, q) \rightarrow q & \perp 0(q, q) \rightarrow q \\ 01(q, q) \rightarrow q & 11(q, q) \rightarrow q & \perp 1(q, q) \rightarrow q \end{array}$$

Automaton for  $X = Y1$ , with final state  $q''$ :

$$\begin{array}{lll} \perp\perp \rightarrow q & 1\perp(q, q) \rightarrow q' & 01(q', q) \rightarrow q'' \\ 00(q'', q) \rightarrow q'' & 00(q, q'') \rightarrow q'' & \end{array}$$

Automaton for  $X = \epsilon$  with final state  $q'$ :

$$\perp \rightarrow q \qquad 1(q, \dots, q) \rightarrow q'$$

Boolean operations and quantifiers taken care of as usual.

Hence the language defined by a WSkS formula can be translated to tree automata.

⇒ WSkS is decidable.

## From tree automata to WSkS

Let  $(\{f_1, \dots, f_n\}, \{q_1, \dots, q_m\}, F, \delta)$  be the given automaton.

The required logical formula is

$$\begin{aligned} \phi(X, X_{f_1}, \dots, X_{f_n}) &\equiv \exists Y_{q_1} \dots \exists Y_{q_m} \cdot \\ &Term(X, X_{f_1}, \dots, X_{f_n}) \wedge Term(Y, Y_{q_1}, \dots, Y_{q_m}) \\ &\wedge \bigvee_{q \in Q_f} \epsilon \in Y_q \\ &\wedge \forall x \bigwedge_{f \in \Sigma, q \in Q} ((x \in X_f \wedge x \in Y_q) \\ &\quad \Rightarrow (\bigvee_{f(q_{i_1}, \dots, q_{i_s}) \rightarrow q \in \delta} \bigwedge_{j=1}^s x_{i_j} \in Y_{q_{i_j}})) \end{aligned}$$

## Program analysis

Prediction of run time behavior of programs.

Application: code optimization techniques used by compilers

$X := -5;$

$A$

while  $X \leq 0$  do

$B$

$X := X + 1;$

$C$

$D$

Program point	Values for $X$
$A$	$\{-5\}$
$B$	$\{-5, \dots, 0\}$
$C$	$\{-4, \dots, 1\}$
$D$	$\{1\}$

Some approximation is unavoidable.

Correct approximation: we overapproximate the set of values taken by a variable at a program point.

## Collecting semantics

$L := cons(a, cons(b, nil));$

$X := c;$

$A$

while ( $L \neq nil$ ) do

$B$

$X := head(L)$

$C$

$L := tail(L)$

$D$

$E$

Point	Environments
$A$	$\{[L \mapsto a.b.nil, X \mapsto c]\}$
$B$	$\left\{ \begin{array}{l} [L \mapsto a.b.nil, X \mapsto c] \\ [L \mapsto b.nil, X \mapsto a] \end{array} \right\}$
$C$	$\left\{ \begin{array}{l} L \mapsto a.b.nil, X \mapsto a \\ [L \mapsto b.nil, X \mapsto b] \end{array} \right\}$
$D$	$\left\{ \begin{array}{l} [L \mapsto b.nil, X \mapsto a] \\ [L \mapsto nil, X \mapsto b] \end{array} \right\}$
$E$	$\{[L \mapsto nil, X \mapsto b]\}$

## Set based analysis

Ignore inter-variable dependencies.

A set variable like  $\mathcal{X}^A$  is used for the set of all possible values of program variable  $X$  at program point  $A$ .

Set constraints are used to describe relationships between these variables.

For statement

$$\begin{array}{l} B \\ X := head(L) \\ C \end{array}$$

we introduce constraints

$$\begin{array}{l} \mathcal{L}^C \supseteq \mathcal{L}^B \\ \mathcal{X}^C \supseteq cons_1^{-1}(\mathcal{L}^B) \end{array}$$

where  $cons_1^{-1}(S) = \{v_1 \mid cons(v_1, v_2) \in S\}$

Solutions of set constraints are sets of trees.

Consider the constraints

$$\mathcal{X} \supseteq a$$

$$\mathcal{X} \supseteq f(f(\mathcal{X}))$$

One possible solution is

$$\mathcal{X} \mapsto \{a, f(a), f(f(a)), \dots\}$$

(The set of all terms)

Another one is

$$\mathcal{X} \mapsto \{a, f(f(a)), f(f(f(f(a))))), \dots\}$$

This is also the least solution (w.r.t set inclusion)

Our interest is in finding least solutions if they exist.

$L := \text{cons}(a, \text{cons}(b, \text{nil}));$

$X := c;$

$A$

while ( $L \neq \text{nil}$ ) do

$B$

$X := \text{head}(L)$

$C$

$L := \text{tail}(L)$

$D$

$E$

$$\mathcal{L}^A \supseteq \text{cons}(a, \text{cons}(b, \text{nil}))$$

$$\mathcal{L}^B \supseteq \mathcal{L}^A \cap \overline{\text{nil}}$$

$$\mathcal{L}^B \supseteq \mathcal{L}^D \cap \overline{\text{nil}}$$

$$\mathcal{L}^C \supseteq \mathcal{L}^B$$

$$\mathcal{L}^D \supseteq \text{cons}_2^{-1}(\mathcal{L}^C)$$

$$\mathcal{L}^E \supseteq \mathcal{L}^A \cap \text{nil}$$

$$\mathcal{L}^E \supseteq \mathcal{L}^D \cap \text{nil}$$

$$\mathcal{X}^A \supseteq c$$

$$\mathcal{X}^B \supseteq \mathcal{X}^A$$

$$\mathcal{X}^B \supseteq \mathcal{X}^D$$

$$\mathcal{X}^C \supseteq \text{cons}_1^{-1}(\mathcal{L}^B)$$

$$\mathcal{X}^D \supseteq \mathcal{X}^C$$

$$\mathcal{X}^E \supseteq \mathcal{X}^A$$

$$\mathcal{X}^E \supseteq \mathcal{X}^D$$

$L := \text{cons}(a, \text{cons}(b, \text{nil}));$

$X := c;$

$A$

while ( $L \neq \text{nil}$ ) do

$B$

$X := \text{head}(L)$

$C$

$L := \text{tail}(L)$

$D$

$E$

$$\mathcal{L}^A \supseteq \text{cons}(a, \text{cons}(b, \text{nil}))$$

$$\mathcal{L}^B \supseteq \mathcal{L}^A \cap \overline{\text{nil}}$$

$$\mathcal{L}^B \supseteq \mathcal{L}^D \cap \overline{\text{nil}}$$

$$\mathcal{L}^C \supseteq \mathcal{L}^B$$

$$\mathcal{L}^D \supseteq \text{cons}_2^{-1}(\mathcal{L}^C)$$

$$\mathcal{L}^E \supseteq \mathcal{L}^A \cap \text{nil}$$

$$\mathcal{L}^E \supseteq \mathcal{L}^D \cap \text{nil}$$

$$\mathcal{X}^A \supseteq c$$

$$\mathcal{X}^B \supseteq \mathcal{X}^A$$

$$\mathcal{X}^B \supseteq \mathcal{X}^D$$

$$\mathcal{X}^C \supseteq \text{cons}_1^{-1}(\mathcal{L}^B)$$

$$\mathcal{X}^D \supseteq \mathcal{X}^C$$

$$\mathcal{X}^E \supseteq \mathcal{X}^A$$

$$\mathcal{X}^E \supseteq \mathcal{X}^D$$

$$\mathcal{L}^A \mapsto \{\text{cons}(a, \text{cons}(b, \text{nil}))\}$$

$$\mathcal{L}^B \mapsto \{\text{cons}(a, \text{cons}(b, \text{nil})), \text{cons}(b, \text{nil})\}$$

$$\mathcal{L}^C \mapsto \{\text{cons}(a, \text{cons}(b, \text{nil})), \text{cons}(b, \text{nil})\}$$

$$\mathcal{L}^D \mapsto \{\text{cons}(b, \text{nil}), \text{nil}\}$$

$$\mathcal{L}^E \mapsto \{\text{nil}\}$$

$$\mathcal{X}^A \mapsto \{c\}$$

$$\mathcal{X}^B \mapsto \{a, b, c\}$$

$$\mathcal{X}^C \mapsto \{a, b\}$$

$$\mathcal{X}^D \mapsto \{a, b\}$$

$$\mathcal{X}^E \mapsto \{a, b, c\}$$

In general we have a set  $\Sigma$  of function symbols of fixed arities.

For each  $f \in \Sigma$  of arity  $n$ , we also have the projection functions  $f_1^{-1}, \dots, f_n^{-1}$ .

Programs consist of assignment statements of the form  $X := t$  together with if-then and while-do statements.

( $t$  is built from function symbols, projection symbols and program variables.)

For each program variable  $X$  and program point  $A$  we have a set variable  $\mathcal{X}^A$ . We generate constraints between the set variables corresponding to the program statements.

A solution  $I$  of the constraints maps each variable  $\mathcal{X}^A$  to some set  $I(\mathcal{X}^A) \subseteq T(\Sigma)$ .

If variable  $X$  ever takes value  $x$  at point  $A$ , then  $x \in I(\mathcal{X}^A)$ . Hence we are interested in the least solution  $I$ .

## Set constraints:

set expressions are of the form

$\mathcal{X}$  (set variable),  $\top$ ,  $\perp$ ,  $f(se_1, \dots, se_n)$ ,  $f_i^{-1}(se)$ ,  $se_1 \cup se_2$ ,  $se_1 \cap se_2$

Given an interpretation  $I$  such that  $I(\mathcal{X}) \subseteq T(\Sigma)$ , we define  $I(se)$  to be:

- $I(\mathcal{X})$  if  $se = \mathcal{X}$
- $T(\Sigma)$  if  $se = \top$
- $\emptyset$  if  $se = \perp$
- $\{f(v_1, \dots, v_n) \mid v_i \in I(se_i)\}$  if  $se = f(se_1, \dots, se_n)$
- $\{v_i \mid f(v_1, \dots, v_n) \in I(se')\}$  if  $se = f_i^{-1}(se')$
- $I(se_1) \cup I(se_2)$  if  $se = se_1 \cup se_2$
- $I(se_1) \cap I(se_2)$  if  $se = se_1 \cap se_2$

Monotonicity property: if  $I_1(\mathcal{X}) \subseteq I_2(\mathcal{X})$  for all  $\mathcal{X}$  then  $I_1(se) \subseteq I_2(se)$ .

We deal with set constraints of the form  $\mathcal{X} \supseteq se$ .

$I$  is a model of this constraint, written  $I \models \mathcal{X} \subseteq se$ , iff  $I(\mathcal{X}) \supseteq I(se)$ .

For a collection  $\mathcal{C}$  of constraints,

$I \models \mathcal{C}$  iff  $I \models \mathcal{X} \supseteq se$  for each  $(\mathcal{X} \supseteq se) \in \mathcal{C}$ .

We write  $I_1 \subseteq I_2$  to mean  $I_1(\mathcal{X}) \subseteq I_2(\mathcal{X})$  for each  $\mathcal{X}$ .

$I_1 \cap I_2$  denotes the interpretation  $I$  such that  $I(\mathcal{X}) = I_1(\mathcal{X}) \cap I_2(\mathcal{X})$ .

If  $I_1, I_2, \dots$ , are each models of  $\mathcal{C}$  then by the monotonicity property  $I_1 \cap I_2 \cap \dots$  is a model of  $\mathcal{C}$ . Hence  $\mathcal{C}$  always has a least model.

First order Horn clauses: another mechanism for describing tree languages.

Tree automata are conveniently described using Horn clauses (Prolog programs)

The clauses

$$q_{\text{even}}(O)$$

$$q_{\text{even}}(S(x)) \Leftarrow q_{\text{odd}}(x)$$

$$q_{\text{odd}}(S(x)) \Leftarrow q_{\text{even}}(x)$$

define the sets of even and odd numbers.

The general idea is to represent automata states using unary predicates of first order logic.

We read atom  $P(t)$  as “tree  $t$  is accepted as state  $P$ ”.

Then clauses of the form  $P(f(x_1, \dots, x_n)) \Leftarrow P_1(x_1) \wedge \dots \wedge P_n(x_n)$  conveniently represent the transitions of usual tree automata.

This clause corresponds to the fact that  $P \in \delta(f, P_1, \dots, P_n)$ .

More general clauses allow us to represent extensions of tree automata.

In general clauses are of the form  $P(t) \Leftarrow P_1(t_1) \wedge \dots \wedge P_n(t_n)$ .

This represents the fact that for any ground substitution  $\sigma$ , if  $t_i\sigma$  is accepted at state  $P_i$  for  $1 \leq i \leq n$  then  $t\sigma$  is accepted at state  $P$ .

We have seen clauses  $P(f(x_1, \dots, x_n)) \Leftarrow P_1(x_1) \wedge \dots \wedge P_n(x_n)$   
(Call them **pop clauses**).

$\epsilon$  transitions are represented using clauses  $P_1(x) \Leftarrow P_2(x)$ .  
(Call them  **$\epsilon$  clauses**).

If now we allow clauses of the form  $P(x) \Leftarrow P_1(x) \wedge \dots \wedge P_n(x)$  we  
obtain **alternating tree automata**.  
(Call these clauses **alternation clauses**).

The other interesting kind of clauses is

$$P(x_i) \Leftarrow Q(f(x_1, \dots, x_n)).$$

These clauses destruct terms instead of constructing them. Call them **push clauses**.

Adding these clauses to (alternating) tree automata produces (alternating) **two-way** tree automata.

(Sometimes also called pushdown systems; but should be distinguished from pushdown automata).

How expressive are these compared to ordinary tree automata?