Counter Systems & Temporal Logics

Lecture 6
Counter systems for data logics

Stéphane Demri
demri@lsv.ens-cachan.fr

LSV, ENS Cachan, CNRS, INRIA

Universidad Nacional de Córdoba, 2010
Plan of the talk

• Previous lectures: counter systems, temporal logics.

• Today’s lecture:
  • Introduction to data logics.
  • Reductions between data logics and verification problems for counter systems.
Models with Data
Ubiquity of data words [Bouyer, IPL 02]

- Data word: \( a_1 \ a_2 \ a_3 \ \cdots \)
  \( d_1 \ d_2 \ d_3 \ \cdots \)
  - Each \( a_i \) belongs to a finite alphabet \( \Sigma \).
  - Each \( d_i \) belongs to an infinite domain \( D \).

- Timed word [Alur & Dill, TCS 94]
  \[
  a \ b \ c \ a \ a \ b \\
  0 \ 0.3 \ 1 \ 2.3 \ 3.5 \ 3.51
  \]

- Runs from counter systems
  \[
  q_0 \ q_2 \ q_3 \ q_2 \ q_3 \ q_2 \\
  0 \ 0 \ 1 \ 2 \ 3 \ 4
  \]

- Integer arrays [Habermehl & Iosif & Vojnar, FOSSACS’08]
  \[
  \]

Explicit relationships with data words and related formalisms in [Alur & Čerňy & Weinstein, CSL’09].
Finite alphabet and infinite domain
Data trees

Extension to data trees (XML documents with values).
[Bojańczyk et al., PODS 06; Jurdiński & Lazić, LICS 07]
Formalisms for Data Words – Temporal Logics
Linear-time temporal operators

\(X\varphi\): next-time \(\varphi\)

\(F\varphi\): sometimes \(\varphi\)

\(G\varphi\): always \(\varphi\)

\(\varphi U \psi\): \(\varphi\) until \(\psi\)
A mechanism for handling data

- Case analyses in formulae are not sufficient with infinite domains.

- A register can store a data value and equality tests are performed between registers and current data values.

- Storing a value in a register:
  \[
  \downarrow_r \varphi \overset{\text{def}}{=} \exists y_r (y_r = x) \land \varphi
  \]

- Equality test between a register and a value: \(\uparrow_r \overset{\text{def}}{=} y_r = x\).
  (in this talk, restriction to the simple equality tests)

- All data values at distinct positions are distinct:
  \[G(\downarrow_r x G \neg \uparrow_r)\]

- Generalization with memory logics, e.g. memory bags have operations “register”, “forget” and “erase”.

[Mera, PhD thesis 09]
Freeze operator

- Freeze quantifier in hybrid logics.
  [Goranko 94; Blackburn & Seligman, JOLLI 95]

- Temporal semantics of imperative programs.
  [Manna & Pnueli, 1992]
  
  Program variable $x$ never decreases below its initial value:
  \[
  \exists y \ (x = y) \land G(x \geq y)
  \]

- Freeze quantifier in real-time logics.
  [Alur & Henzinger, JACM 94]
  
  $y \cdot \varphi(y)$ binds the variable $y$ to the current time $t$.

- Predicate $\lambda$-abstraction [Fitting, JLC 02].
  \[
  \langle y \cdot F P(y) \rangle(c) : \text{current value of constant } c \text{ satisfies the predicate } P.
  \]

- See also description logics over concrete domains.
  [Baader & Hanschke, IJCAI’91; Lutz, TOCL 04]
Hybrid logics as data logics

- Most standard models for modal logics are graphs in which nodes are labelled by propositional valuations.

- For a given formula, the set of propositional valuations is a finite alphabet.

- $\downarrow y \varphi$: $\varphi$ holds true in the variant model where proposition $y$ is true only at the current state.

  [Goranko 94; Blackburn & Seligman, JOLLI 95].

- Models are enriched with node adresses.

- “Every reachable state can be visited infinitely often”: $\text{AG } \downarrow y \text{ E } XF y$

![Diagram of a graph with nodes and arrows showing the relationship between reachable states.](attachment:diagram.png)
LTL with registers: $\text{LTL} \downarrow$

- **LTL$\downarrow$ formulae:**

  $\varphi ::= a | \uparrow_r | \neg \varphi | \varphi \land \varphi | \varphi \lor \varphi | \varphi U \varphi | X \varphi | \downarrow_r \varphi$

  where $a \in \Sigma$ and $r \in \mathbb{N}^+$. 

- Register valuation $f$: finite partial map from $\mathbb{N}^+$ to $\mathbb{N}$ ($= D$).

- Models: finite or infinite data words over the alphabet $\Sigma$.

- Satisfaction relation:

  $$\sigma, i \models_f \uparrow_r \Leftrightarrow r \in \text{dom}(f) \text{ and } f(r) = d_i$$

  $$\sigma, i \models_f \downarrow_r \varphi \Leftrightarrow \sigma, i \models f[r \rightarrow d_i] \varphi$$

  ($d_i$: data value at position $i$)

- Unlike standard LTL, $\text{LTL} \downarrow$ can store a data value and perform equality tests.
Examples

- Nonce property: $G(\downarrow_1 XG\neg \uparrow_1)$.

\[ \downarrow_1 X \uparrow_1 \approx x = Xx \]

\[
\begin{array}{ccccccc}
  a & a & b & d & a & b & 0 \\
\end{array}
\]

$\not\models F(a \land \downarrow_1 XF(a \land \uparrow_1))$
An another view on LTL ↓

- Standard LTL models are of the form \( \mathbb{N} \to \mathcal{P}(\text{PROP}) \) for some countably infinite set \( \text{PROP} \) of atomic propositions.

- An LTL formula \( \varphi \) built over \( \{p_1, \ldots, p_k\} \) constrains the models only for \( \{p_1, \ldots, p_k\} \)

- No LTL formula characterizes the class of models for which any two distinct positions have distinct valuations.

- \( \text{LTL} \downarrow = \text{extension of LTL (with standard models) where the registers store valuations in } \mathcal{P}(\text{PROP} \setminus \text{PROP}_k) \) and the alphabet is \( \mathcal{P}(\text{PROP}_k) \) with \( \text{PROP}_k = \{p_1, \ldots, p_k\} \).
Complexity of satisfiability problems

- Finitary and infinitary satisfiability problem for LTL are \( \text{PSPACE}\)-complete. [Sistla & Clarke, JACM 85]

- What about \( \text{LTL} \downarrow \) with one register, with all registers etc.?

- Infinitary satisfiability problem for \( \text{LTL} \downarrow \) restricted to \( X \) and \( F \) and to a single register is undecidable.

- Finitary satisfiability problem for \( \text{LTL} \downarrow \) restricted to a single register is decidable but nonprimitive recursive. [Demri & Lazić, TOCL 09]

- Finitary satisfiability problem for \( \text{LTL} \downarrow \) restricted to \( F \) and
  - to a single register is nonprimitive recursive too.
  - to two registers is undecidable. [Figueira & Segoufin, MFCS’09]

- Nonprimitive recursiveness uses [Schnoebelen, IPL 02].
How Counter Systems Enter into the Play
Counter automata (CA)

- Counter system = finite-state automaton + counters.
- Counter: program variable interpreted by a non-negative integer.

Counter automaton $S = (Q, n, \delta)$
- Finite set of control states $Q$.
- Transitions in $\delta \subseteq Q \times \{\text{zero}(i), \text{inc}(i), \text{dec}(i) : i \in [1, n]\} \times Q$.
- Dimension $n$ (number of counters).

Runs of the form

$$\rho = q_0, \vec{x}_0 \rightarrow q_1 (\in Q), \vec{x}_1 (\in \mathbb{N}^n) \rightarrow q_2, \vec{x}_2 \rightarrow \ldots$$
Reachability problems

- Reachability problem:
  \textbf{Input:} counter automaton $S$, $(q, 0)$ and $(q', 0)$.
  \textbf{Question:} is $(q, 0) \xrightarrow{*} (q', 0)$?

- Control state reachability problem:
  \textbf{Input:} counter automaton $S$, $(q, 0)$ and $q'$.
  \textbf{Question:} is $(q, 0) \xrightarrow{*} (q', \vec{x}')$ for some $\vec{x}'$?

- Control state repeated reachability problem:
  \textbf{Input:} counter automaton $S$, $(q, 0)$ and $q_f$.
  \textbf{Question:} is there an infinite run from $(q, \vec{x})$ such that $q_f$ is repeated infinitely often?

- Covering problem (extending control state reachability):
  \textbf{Input:} counter automaton $S$, $(q, 0)$ and $(q', \vec{x}')$.
  \textbf{Question:} is $(q, 0) \xrightarrow{*} (q', \vec{x}'')$ with $\vec{x}' \preceq \vec{x}''$?
  ($\preceq$ is defined pointwise)
Counter automata generate data words

- A counter automaton and an initial configuration generate a set of runs viewed as data words with multiple data values.
- The finite alphabet is $Q$.
- Extension of freeze operators to $\downarrow_r^j$ and $\uparrow_r^j$ with $j \in [1, n]$. 
A counter stores a single natural number.

A Minsky machine can be viewed as a deterministic finite-state automaton with two counters.

Operations on counters:
- Check whether the counter is zero.
- Increment the counter by one.
- Decrement the counter by one if nonzero.

Halting problem (≈ control state reachability problem):
- **input:** a Minsky machine $M$;
- **question:** is the unique computation halts?

The halting problem is undecidable and Minsky machines are Turing-complete.

[Minsky, 67]
Reachability Problems for Gainy CA
Gainy counter automata

- Faulty systems perform errors such as losses or gains, e.g., see works on lossy channel systems.  
  [Abdulla & Jonsson, IC 96]

- Three ways to model gainy counter automata:
  1. Standard CA \((Q, n, \delta)\) such that for \(q \in Q\) and \(i \in [1, n]\), \(q^{\text{inc}(i)} \xrightarrow{} q \in \delta\).
  
  2. Alternative one-step relation: \((q, \vec{x}) \xrightarrow{t} g (q', \vec{x'})\) iff there are \(\vec{y}, \vec{y}'\) in \(\mathbb{N}^n\) such that

     \[
     \vec{x} \preceq \vec{y} \quad \text{and} \quad (q, \vec{y}) \xrightarrow{t} (q', \vec{y}') \quad \text{(exact step)} \quad \text{and} \quad \vec{y}' \preceq \vec{x'}
     \]

  3. Gains occur in a lazy way: decrement on zero has no effect.
Benefits from Gainy CA

- Features:
  - Increment, decrement and zero-test.
  - Incrementation errors.

- Control state reachability problem is decidable but with a nonprimitive recursive complexity.
  
  See e.g., [Urquhart, JSL 99; Schnoebelen, IPL 02]

- Control state repeated reachability problem is undecidable.
  
  [Demri & Lazić, TOCL 09]
  
  (adapt a proof from [Ouaknine & Worrell, FOSSACS’06])

- These problems reduce to corresponding satisfiability problems for $\text{LTL}^\downarrow$ restricted to $X$ and $F$ and to a single register.
Simulating Gainy CA

- Gainy CA $S$ with initial configuration $(q_0, \vec{0})$.

- For $t \in \delta$, $\Sigma(t)$ denotes the instruction labelling it in $\Sigma = \{\text{inc}(i), \text{dec}(i), \text{zero}(i) : i \in [1, n]\}$.

- Let us build $\varphi$ in $\text{LTL}^\downarrow$ s.t. $\varphi$ is satisfiable iff $(S, (q_0, \vec{0}))$ has an infinite run with $q_f$ occurring infinitely often.

- $\varphi$ is satisfiable only in models in which each position is labelled by a transition and by a value in $\mathbb{N}$.

- Infinite models of $\varphi$ are of the form $(t_0, y_0), (t_1, y_1), (t_2, y_2), \cdots$ with $t_i \in \delta$ and $y_i \in \mathbb{N}$.

- For $I, J \in \mathbb{N}$, $I \sim J \iff y_I = y_J$. 

Simulating gainy CA (II)

- Let us explain how the run from \((q_0, \vec{0})\) below is encoded.

\[
(q_0, \vec{x}_0) \xrightarrow{a_0} (q_1, \vec{x}_1) \xrightarrow{a_1} \cdots \xrightarrow{a_{K-1}} (q_K, \vec{x}_K) \cdots
\]

- Projection of the model over \(\delta\) is

\[
t_0 t_1 t_2 \cdots = q_0 \xrightarrow{a_0} q_1, q_1 \xrightarrow{a_1} q_2, \cdots
\]

and \(q_f\) is repeated infinitely often.

- Initial state is \(q_0\):

\[
\bigwedge t \quad t=q_0 \xrightarrow{a} q
\]

- The sequence of transitions respects \(\delta\):

\[
G(\bigwedge t : (t \Rightarrow \bigvee t'))
\]

\[
t=q \xrightarrow{a} q' \in \delta \quad t'=q' \xrightarrow{a} q''
\]
Simulating Gainy CA (III)

- Control state $q_f$ is visited infinitely often: $\exists t, q \xrightarrow{a} q_f$

- Each increment or decrement is associated to a unique value.

- For $a \in \Sigma$, $a$ is also used as a shortcut for $\exists t, q \xrightarrow{b} q' \in \delta, a=b$

- For $i, j \in [1, n]$, there are no two positions for increments [resp. decrements] having the same value:

  $$G(\text{inc}(i) \Rightarrow \neg (\downarrow_1 X F (\uparrow_1 \land \text{inc}(j)))) \land G(\text{dec}(i) \Rightarrow \neg (\downarrow_1 X F (\uparrow_1 \land \text{dec}(j))))$$
Simulating Gainy CA (IV)

- The two next conditions are formulated in such a way to avoid using the until operator $\mathcal{U}$.

- For $i \in [1, n]$ and $J > I$, if $\Sigma(t_I) = \text{inc}(i)$ and $\Sigma(t_J) = \text{zero}(i)$, then there is no $K > J$ such that $\Sigma(t_K) = \text{dec}(i)$ and $I \sim K$:

  $$G(\text{inc}(i) \Rightarrow \downarrow_1 \neg (F(\text{zero}(i) \land (F(\uparrow_1 \land \text{dec}(i))))))$$

- For $i \in [1, n]$, if there are $J > I$ such that $\Sigma(t_I) = \text{inc}(i)$ and $\Sigma(t_J) = \text{zero}(i)$, then there is $K > I$ such that $\Sigma(t_K) = \text{dec}(i)$ and $I \sim K$.

  $$G((\text{inc}(i) \land F \text{ zero}(i)) \Rightarrow \downarrow_1 (F(\text{dec}(i) \land \uparrow_1)))$$

- $\varphi$ is satisfiable iff $(S, (q_0, \vec{0}))$ has an infinite run such that $q_f$ occurs infinitely often.
Gainy CA for $\text{LTL} \downarrow$ with one register!

- Control state repeated reachability problem for Gainy CA can be reduced to infinitary satisfiability for $\text{LTL} \downarrow$ restricted to one register. $\rightarrow$ undecidability

- Control state reachability problem for Gainy CA can be reduced to finitary satisfiability for $\text{LTL} \downarrow$ restricted to one register. $\rightarrow$ nonprimitive recursiveness

- In the finitary case, there is a converse reduction.
About nonprimitive recursiveness

- Control state reachability problem for gainy CA is nonprimitive recursive.
  See e.g., [Urquhart, JSL 99; Schnoebelen, MFCS’10]

- Ackermann function is nonprimitive recursive.
  (grows faster than any primitive recursive function)

- Decidable nonclassical logics with nonprimitive recursive complexity:
  - Products of modal logics with expanding domains by reduction from the reachability problem for lossy channel systems.
    [Gabelaia et al., APAL 06]
  - Relevance logic LR+ (and fragments) by introducing a branching extension of CA.
    [Urquhart, JSL 99]
  - Finitary Metric Temporal Logic MTL.
    [Ouaknine & Worrell, LICS’05]
Decidable logic \( CLTL^\diamond \) with repeating values

- Formulae:

\[
\varphi ::= x = x^i y \mid x = \Diamond y \mid \varphi \land \varphi \mid \neg \varphi \mid X \varphi \mid \varphi \cup \varphi \mid Y \varphi \mid \varphi S \varphi
\]

- Finitary and infinitary satisfiability for \( CLTL^\diamond \) is decidable.
  
  
  [Demri & D’Souza & Gascon, LFCS’07]
  - By reduction to checking fairness conditions in Petri nets.
    [Jančar, TCS 90]
  - \( \text{PSPACE} \)-completeness with a unique flexible variable.
  - Decidability is preserved with MSO-definable operators.
Model-Checking Counter Automata
Motivations

1. Model-checking with focus on data values

To analyze runs of operational models with focus on data values (beyond control state reachability).
E.g., “there is a value of counter 1 such that infinitely often counter 2 takes that value iff infinitely often counter 3 takes that value”:

\[ F \downarrow^1 (G F \uparrow^2 \leftrightarrow G F \uparrow^3) \]


Current instance:

- Operational models: classes of counter automata for which the reachability problem is decidable.

- Most often, the reachability sets are definable in Presburger arithmetic (decidable first-order theory of \((\mathbb{N}, +)\)).

- Specification language: LTL with registers.
Model-checking counter automata

- Infinitary model-checking problem $MC^\omega(LTL\downarrow)$:
  
  **Input:** CA $S = (Q, n, \delta)$, configuration $(q, \vec{x}) \in Q \times \mathbb{N}^n$, and a sentence $\varphi \in LTL\downarrow$ over alphabet $Q$;
  
  **Question:** Is there an infinite run $\rho$ such that $\rho, 0 \models \varphi$?

- Undecidability for nondeterministic one-counter automata:
  
  - $MC^{<\omega}(LTL_1\downarrow(X, F))$ is $\Sigma^0_1$-complete.
  
  - $MC^\omega(LTL_1\downarrow(X, F))$ is $\Sigma^1_1$-complete.

- Reachability sets are semilinear but universal problem for one-counter automata with alphabet is undecidable.

[Ibarra, MST 79]
Reversal-bounded counter automata

- Reversal: Alternation from nonincreasing mode to nondecreasing mode and vice-versa.

- Sequence with 3 reversals:
  
  001122334444333222233344445555555

- Reversal-bounded counter automata: each run has a bounded number of reversals. [Ibarra, JACM 78]

- Reachability sets are effectively Presburger-definable. [Ibarra, JACM 78]

- Control state repeated reachability problem is decidable. [Dang & Ibarra & San Pietro, FST&TCS’01]
Vector addition systems with states (VASS)

- Succinct CA without zero-tests.
- Transitions of the form $q \xrightarrow{\vec{b}} q'$ with $\vec{b} \in \mathbb{Z}^n$, which is a shortcut for $\bigwedge_{i \in [1,n]} x'_i = x_i + \vec{b}(i)$.
- $\approx$ Petri nets (models of greater practical appeal).
- The reachability problem is decidable.

[Kosaraju, STOC’82; Mayr, SIAM 84]
Towards flatness

- $\text{MC}^\omega(\text{LTL} \downarrow)$ restricted to reversal-bounded CA and to formulae with at most one register is undecidable.

- Flat formulae: positive occurrence of $\varphi_1 \lor \varphi_2$ implies $\downarrow$ does not occur in $\varphi_1$.

- $\neg (q \lor \downarrow^1 \varphi)$ is not a flat formula.

- Flatness is a standard means to regain decidability for memoryful linear-time temporal logics.
Introducing parameters

- $\text{MC}^\omega(LTL_{↓})$ restricted to reversal-bounded CA and to flat formulae is decidable. [Demri & Sangnier, FOSSACS’10]

- Decidability proof uses that the control state repeated reachability problem for parameterized reversal-bounded CA is decidable. [Ibarra et al., TCS 02]

- Transitions of the form $\textit{add}(z)$ with parameter $z$.

- Reachability questions are relative to parameter values.
Formalisms for Data Words – First-Order Logics
First-order logic on data words

- Data word: nonempty finite sequence of pairs from $\Sigma \times \mathbb{N}$.

- Variable valuation $v$ for a model $\sigma$: map from VAR to the positions of $\sigma$.

- Variables are interpreted as positions.

- Formulae of the logic $\text{FO}^{\Sigma}(\sim, <, +1)$ ($\Sigma$ is a finite alphabet)

$$\varphi ::= a(x) \mid x \sim y \mid x < y \mid x = y + 1 \mid \neg \varphi \mid \varphi \land \varphi \mid \exists x \varphi$$

- Last position is labelled by the letter $a \in \Sigma$:

$$\exists x \ (\neg \exists y \ x < y) \land a(x)$$
Data words as first-order structures

- Alphabet $\Sigma = \{a_1, \ldots, a_N\}$ and infinite domain $\mathbb{N}$.

- Data word $\sigma = (a_{i_1}, d_1) \cdots (a_{i_K}, d_K)$ is equivalent to
  $$(\{1, \ldots, K\}, <, \sim, +1, P_1, \ldots, P_N)$$

- For $j, j' \in \{1, \ldots, K\}$, $j \sim j'$ iff $d_j = d_{j'}$.

- For $l \in \{1, \ldots, N\}$, $P_l \overset{\text{def}}{=} \{j \in \{1, \ldots, K\} : a_{ij} = a_l\}$.

- First-order logic can be naturally interpreted over such structures.
\[
\sigma \models_v a(x) \overset{\text{def}}{\iff} \Sigma(x) = a \quad \text{(letter at position } x) \\
\sigma \models_v x \sim y \overset{\text{def}}{\iff} \mathbb{N}(x) = \mathbb{N}(y) \quad \text{(data at positions)} \\
\sigma \models_v x < y \overset{\text{def}}{\iff} v(x) < v(y) \\
\sigma \models_v x = y + 1 \overset{\text{def}}{\iff} v(x) = v(y) + 1 \\
\sigma \models_v \exists x \varphi \overset{\text{def}}{\iff} \text{there is position } i \text{ s.t. } \sigma \models_{v[x\mapsto i]} \varphi.
\]
Theorem: Satisfiability problem for \( \text{FO}_2(\sim, <, +1) \) is decidable. [Bojańczyk et al., LICS 06]

Proof in two steps:
- Satisfiability is first reduced to nonemptiness for data automata (undefined herein).
- Nonemptiness for data automata is then reduced to the reachability problem for VASS.

Theorem: There is a reduction from the reachability problem for VASS to finitary satisfiability for \( \text{FO}_2(\sim, <, +1) \).
Fixing a few more things (proof)

- Instance: \( S = (Q, n, \delta), (q_i, \vec{0}), (q_f, \vec{0}) \).

- \( \Sigma = Q \uplus \{ \text{inc}(i), \text{dec}(i) : i \in [1, n] \} \).
  (below \( a \in \{ \text{inc}(i), \text{dec}(i) : i \in [1, n] \} \))

- The run \((q_0, \vec{x}_0) \overset{a_0}{\rightarrow} (q_1, \vec{x}_1) \overset{a_1}{\rightarrow} \cdots \overset{a_{K-1}}{\rightarrow} (q_K, \vec{x}_K)\) encoded by a data word with projection \(q_0 a_0 q_1 a_1 \cdots a_{K-1} q_K\).

- Run

\[
\begin{array}{cccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 \\
(0) & (1) & (2) & (2) & (1) & (0) & (0)
\end{array}
\]

corresponds to data word

\[
\begin{array}{cccccccc}
q_0 & \text{inc}(1) & q_1 & \text{inc}(1) & q_2 & \text{inc}(2) & q_3 & \text{dec}(1) & q_4 & \text{dec}(1) & q_5 & \text{dec}(2) & q_6 \\
* & k_1 & * & k_2 & * & k_3 & * & k_1 & * & k_2 & * & k_3 & *
\end{array}
\]
Enforcing the projection

- $\varphi_{proj}$: conjunction of the formulae below.
  - The first letter is $q_i$:
    \[ \exists x (\neg \exists y \ y < x) \land q_i(x) \]
  - The last letter is $q_f$:
    \[ \exists x (\neg \exists y \ x < y) \land q_f(x) \]
  - Sequence of locations/actions respects graph of $S$:
    \[
    \forall x \left( \bigvee_{q \in Q} q(x) \right) \Rightarrow \left( (\neg \exists y \ x < y) \lor \bigvee_{q^a \rightarrow q' \in \delta} \left( q(x) \land (\exists y \ y = x + 1 \land a(y)) \land \right) q'(x) \right) 
    \]
- Observe the nice (and standard) recycling of variables.
Constraints on data values

• To encode counter values, each increment or decrement is attached to a datum.

• A desirable data word:

\[ q_0 \text{ inc}(1) q_1 \text{ inc}(1) q_2 \text{ inc}(2) q_3 \text{ dec}(1) q_4 \text{ dec}(1) q_5 \text{ dec}(2) q_6 \]
\[ \star k_1 \star k_2 \star k_3 \star k_1 \star k_2 \star k_3 \star \]

• \( \varphi \): conjunction of \( \varphi_{proj} \) and formulae below.

- For \( i, j \in [1, n] \), there are no two positions labelled by \( \text{inc}(i) \) and \( \text{inc}(j) \) having the same datum:

\[
\forall x \ y \ (x < y \land \text{inc}(i)(x) \land \text{inc}(j)(y)) \Rightarrow \neg(x \sim y).
\]

(remember \( \text{inc}(i) \) and \( \text{dec}(i) \) are also letters in \( \Sigma \))

- Same with \( \text{dec}(i) \) and \( \text{dec}(j) \):

\[
\forall x \ y \ (x < y \land \text{dec}(i)(x) \land \text{dec}(j)(y)) \Rightarrow \neg(x \sim y).
\]
Constraints on data values (II)

- For \( i \in [1, n] \), for every position labelled by \( \text{dec}(i) \), there is a past position labelled by \( \text{inc}(i) \) with the same data value:

\[
\forall x \ \text{dec}(i)(x) \implies (\exists y \ (y < x) \land (x \sim y) \land \text{inc}(i)(y))
\]

- Since in the final configuration, any counter value is zero, we impose that for \( i \in [1, n] \), for every position labelled by \( \text{inc}(i) \), there is a future position labelled by \( \text{dec}(i) \) with same data value:

\[
\forall x \ \text{inc}(i)(x) \implies (\exists y \ (x < y) \land (x \sim y) \land \text{dec}(i)(y))
\]

- One can show \((q_f, \vec{0})\) is reachable from \((q_i, \vec{0})\) iff \( \varphi \) is satisfiable.
FO3(∼, <, +1) is undecidable
[Bojańczyk et al., LICS 06]

- Extend VASS with zero-tests.
- Nonemptiness problem (or equivalent control state reachability) is undecidable.
- Use the third variable to encode zero-tests:

\[ \forall x \; \text{zero}(i)(x) \Rightarrow \]

\[ (\forall y \; (y < x \land \text{inc}(i)(y)) \Rightarrow \exists z \; ((y < z < x) \land \text{dec}(i)(y) \land (y \sim z))) \]
Formalisms for Data Words – Automata
Specifying classes of data words

- Register automata
  - Register automata  [Kaminski & Francez, TCS 94]
  - Data automata  [Bouyer & Petit & Thérien, IC 03]
  - Machines for strings over infinite alphabets.  [Neven & Schwentick & Vianu, TOCL 04]
  - See the survey  [Segoufin, CSL’06]

- Many new classes
  - Class automata [Bojańczyk & Lasota, LICS’10].
  - Variable automata.  [Grumberg & Kupferman & Sheinval, LATA’10]
  - etc.

- Many other formalisms
  - Rewriting systems with data  [Bouajjani et al., FCT’07]
  - Hybrid logics  [Schwentick & Weber, STACS’07]
  - XPath on data trees.
Other relationships with counter automata

- Class counting automata counts the number of data values along a word.\[\text{[Manuel & Ramanujan, RP’09]}\]
  - Comparisons to constant values.
  - Nonemptiness reduces to the covering problem for VASS.
  - EXPSPACE upper bound with integers in unary.

- Automata for bounded-depth data trees:
  - Decidability of nonemptiness problem by reduction into priority CA.\[\text{[Björklund & Bojańczyk, ICALP’07]}\]
  - Nonemptiness for priority CA shown decidable in [Reinhardt, Hab. thesis 05].

- Safety fragment of $\mathsf{LTL}^\downarrow$ with one register on infinite data words.\[\text{[Lazić, FST&TCS’06]}\]
  - No $\mathsf{U}$-subformulae with positive polarity.
  - EXPSPACE upper bound by introducing a subclass of gainy counter automata.
What about languages?

- Each formula defines a class of data words (those satisfied by the formula) and a class of words over a finite alphabet obtained by projection.

- Each data logic defines a class of languages made of words over a finite alphabet.

- Each counter automaton (with alphabet and augmented with initial and final control states) defines a class of languages made of words over a finite alphabet.

- \(\text{LTL} \downarrow\) with a unique register is equivalent to gainy counter automata. \([\text{Demri & Lazić, TOCL 09}]\)

- \(\text{FO}_2(\sim, <, +1)\) is equivalent to VASS. \([\text{Bojańczyk et al., LICS’06}]\)

- See new relationships in [Bojańczyk & Lasota, LICS’10].
Perspectives
Branching extensions

- Data trees
  - See e.g., register automata for data trees in
    [Jurdziński & Lazić, LICS’07; Figueira, PODS’09]

- Branching VASS: computations are finite trees instead linear structures. [Verma & Goubault-Larrecq, DTMCS 05]
  - Reachability problem for BVASS can be reduced to satisfiability for FO2 over data trees.
    [Bojańczyk et al., PODS’06]

- Decidability status of the reachability problem is open.

- Covering problem and boundedness problem are $2\text{EXP\textsc{Time}}$-complete. [Demri et al., FST&TCS’09]
Well-structured transition systems

- Well-structured transition system \((S, \rightarrow, \leq)\)
  [Finkel & Schnoebelen, TCS 01]
  - \(\leq\) is a well-quasi-ordering: for any infinite sequence \(\vec{x}_0, \vec{x}_1, \ldots\) in \(S\), there are \(i < j\) such that \(\vec{x}_i \leq \vec{x}_j\).
  - \(\rightarrow\) and \(\leq\) are upward compatible:

    ![Diagram](image.png)

- Most decidability proofs uses the well-structuredness of underlying transition systems.
- This is made explicit in [Figueira, ICDT’10].
- How far can we use well-structured transition systems to show decidability? What about computational complexity?
Conclusion

- Data logics/automata can benefit from counter systems.
- Verification of counter systems need data logics as specification languages.
- Relationships with other nonclassical logics: product of modal logics, relevance logics, memoryful temporal logics, hybrid logics, memory logics . . .
- Many open problems in decidability, complexity, expressive power of data logics.
  
  See e.g., recent [David & Libkin & Tan, LPAR’10]

- Study of new automata models:
  - Variable automata.
  [Grumberg & Kupferman & Sheinvald, LATA’10]
  - Register automata with guess and spread.
  [Figueira, ICDT’10]