How Good are Counter Systems for Data Logics

Stéphane Demri

NYU & CNRS – Marie Curie Fellow

May 2013
Models with Data
Ubiquity of Data Words [Bouyer, IPL 02]

- Data word: \( a_1 \ a_2 \ a_3 \ \ldots \)
  - Each \( a_i \) belongs to a finite alphabet \( \Sigma \).
  - Each \( \delta_i \) belongs to an infinite domain \( D \).

- Timed word [Alur & Dill, TCS 94]
  \[
  \begin{array}{cccccccc}
  a & b & c & a & a & a & a & b \\
  0 & 0.3 & 1 & 2.3 & 3.5 & 3.51 \\
  \end{array}
  \]

- Runs from counter systems
  \[
  \begin{array}{cccccccc}
  q_0 & q_2 & q_3 & q_2 & q_3 & q_2 \\
  0 & 0 & 1 & 2 & 3 & 4 \\
  \end{array}
  \]

- Integer arrays [Habermehl & Iosif & Vojnar, FOSSACS’08]
  \[
  \begin{array}{cccccccc}
  \end{array}
  \]

Explicit relationships with data words and related formalisms in [Alur & Čeřný & Weinstein, CSL’09].
Finite Alphabet and Infinite Domain

URL_1 URL_2 URL_1 URL_2 URL_3 URL_3

a a b d a b

3 2.5 3 2.5 4 4

Models with Data
Formalisms for Data Words – Temporal Logics
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 equality tests)

- All data values at distinct positions are distinct:

  \[ G(\downarrow_r XG\neg \uparrow_r) \]
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].
  $\langle y \cdot F \ P(y) \rangle(c)$: current value of constant $c$ satisfies the predicate $P$. 
LTL with Registers: $\text{LTL} \downarrow$

- **LTL$\downarrow$** formulae:
  \[
  \varphi ::= a \mid \uparrow r \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \mathbin{U} \varphi \mid X \varphi \mid \downarrow r \varphi
  \]
  where $a \in \Sigma$ and $r \in \mathbb{N}^+$. 

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

- Models: finite/infinite data words $\varnothing w$ over the alphabet $\Sigma$.

- Satisfaction relation:
  \[
  \varnothing w, i \models_f \uparrow r \overset{\text{def}}{\iff} r \in \text{dom}(f) \text{ and } f(r) = \varnothing_i \\
  \varnothing w, i \models_f \downarrow r \varphi \overset{\text{def}}{\iff} \varnothing w, i \models_{f[r\mapsto \varnothing_i]} \varphi
  \]
  ($\varnothing_i$: 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}{cccc}
a & a & b & d \\
a & b & a & b \\
\end{array}
\)

, $0 \not\models F(a \land \downarrow_1 XF(a \land \uparrow_1))$
Complexity of Satisfiability Problems

- Finitary and infinitary satisfiability problem for LTL are PSPACE-complete. [Sistla & Clarke, JACM 85]

- What about $\text{LTL}^↓$ with one register, with all registers etc.?

- Infinitary satisfiability problem for $\text{LTL}^↓$ restricted to $X$ and $F$ and to a single register is undecidable.

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

- Finitary satisfiability problem for $\text{LTL}^↓$ 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 automaton $\mathcal{C} = (Q, n, \delta)$
  - 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: $\rho = q_0 \xrightarrow{\vec{x}_0} q_1 (\in Q) \xrightarrow{\vec{x}_1 (\in \mathbb{N}^n)} q_2 \xrightarrow{\vec{x}_2} \ldots$
Reachability Problems

- **Reachability problem:**
  
  **Input:** counter automaton $C$, $(q_i, \vec{0})$ and $(q_f, \vec{0})$.
  
  **Question:** is $(q_i, \vec{0}) \xrightarrow{*} (q_f, \vec{0})$?

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

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

- **Covering problem (extending control state reachability):**
  
  **Input:** counter automaton $C$, $(q_i, \vec{0})$ and $(q_f, \vec{x}_f)$.
  
  **Question:** is $(q_i, \vec{0}) \xrightarrow{*} (q_f, \vec{x})$ with $\vec{x}_f \preceq \vec{x}$?
  
  ($\preceq$ is defined pointwise)
Turing-completeness of Minsky Machines

- A counter stores a single natural number.

- A Minsky machine is 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:** does the unique computation halt?

- 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 \(i \in [1, n]\),
     \[
     q \xrightarrow{\text{inc}(i)} q' \in \delta.
     \]
  2. Alternative one-step relation: \((q, \vec{x}) \xrightarrow{t} (q', \vec{x}')\) 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 \( C \) 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 \((C, (q_0, \vec{0}))\) 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 \((t_0, \vartheta_0), (t_1, \vartheta_1), (t_2, \vartheta_2), \ldots \) with \( t_i \in \delta \) and \( \vartheta_i \in \mathbb{N} \).

• For \( I, J \in \mathbb{N} \), \( I \sim J \overset{\text{def}}{\iff} \vartheta_I = \vartheta_J \).
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 \(q_0\):

\[
\bigvee_t t = q_0 \xrightarrow{a} q
\]

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

\[
G( \bigwedge_{t=q \xrightarrow{a} q' \in \delta} (t \Rightarrow X \bigvee t'))
\]

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

- Control state $q_f$ is visited infinitely often: $\text{GF} \bigvee_{t=q^{a\rightarrow}q_f} t$
- Each increment or decrement is associated to a unique value.

\[
\begin{align*}
\text{inc}(1) & \quad \text{inc}(1) & \quad \text{dec}(1) & \quad \text{inc}(1) & \quad \text{dec}(1) & \quad \text{dec}(1) & \quad \text{zero}(1)
\end{align*}
\]

- For $a \in \Sigma$, $a$ is also used as a shortcut for $t=q^{b\rightarrow}q' \in \delta, a=b$
- For all $i, j \in [1, n]$, there are no two positions for increments [resp. decrements] having the same value:

\[
\begin{align*}
G(\text{inc}(i) & \Rightarrow \neg(\downarrow_1 \text{XF}(\uparrow_1 \land \text{inc}(j)))) & \quad G(\text{dec}(i) & \Rightarrow \neg(\downarrow_1 \text{XF}(\uparrow_1 \land \text{dec}(j))))
\end{align*}
\]
Simulating Gainy CA (IV)

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

- For every $i \in [1, n]$ 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 every $i \in [1, n]$, 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 $(C, (q_0, \vec{0}))$ 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.
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 $\nu$: map from $\text{VAR}$ to the positions of data word $\omega$.
- 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 $\varnothing w = (a_{i_1}, \varnothing_1) \cdots (a_{i_K}, \varnothing_K)$ is equivalent to
  $$(\{1, \ldots, K\}, <, \sim, +1, P_1, \ldots, P_N)$$

  - For all $j, j' \in [1, K]$, $j \sim j'$ iff $\varnothing_j = \varnothing_{j'}$.

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

- First-order logic can be naturally interpreted over such structures.
\[ \mathcal{v} \models \mathcal{a}(x) \iff \sum(x) = a \quad \text{(letter at position } x) \]

\[ \mathcal{v} \models x \sim y \iff \mathcal{N}(x) = \mathcal{N}(y) \quad \text{(data at positions)} \]

\[ \mathcal{v} \models x < y \iff \nu(x) < \nu(y) \]

\[ \mathcal{v} \models x = y + 1 \iff \nu(x) = \nu(y) + 1 \]

\[ \mathcal{v} \models \exists x \varphi \iff \text{there is position } i \text{ s.t. } \mathcal{v} \models _{[x \mapsto i]} \varphi. \]
FO2 and VASS

• **Theorem:** Satisfiability problem for FO2(∼, <, +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 FO2(∼, <, +1).
Fixing a Few More Things (proof)

• Instance: \( C = (Q, n, \delta), (q_i, \bar{0}), (q_f, \bar{0}) \).

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

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

• Run

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

corresponds to data word

\[
\begin{array}{ccccccccccc}
q_0 & \text{inc}(1) & q_1 & \text{inc}(1) & q_2 & \text{inc}(2) & q_3 & \text{dec}(1) & q_4 & \text{dec}(1) & q_5 & \text{dec}(2) \\
* & \odot_1 & * & \odot_2 & * & \odot_3 & * & \odot_1 & * & \odot_2 & * & \odot_3
\end{array}
\]
Enforcing the Projection

- $\varphi_{\text{proj}}$: 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 $C$:
    \[ \forall x \, (\bigvee_{q \in Q} q(x)) \Rightarrow ((\neg \exists y \, x < y) \land \bigvee_{q \rightarrow q' \in \delta} (q(x) \land (\exists y \, y = x + 1 \land a(y))) \land (\exists y \, y = x + 1 \land (\exists x \, x = y + 1 \land q'(x)))))) \]
  
  - 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:

\[
q_0 \text{ inc}(1) q_1 \text{ inc}(1) q_2 \text{ inc}(2) q_3 \text{ dec}(1) q_4 \text{ dec}(1) q_5 \text{ dec}(2)
\]

\[
\ast \ d_1 \ast \ d_2 \ast \ d_3 \ast \ d_1 \ast \ d_2 \ast \ d_3
\]

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

- For all \(i, j \in [1, n]\), 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:

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

- 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:

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

- 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:

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