Reasoning about sequences of memory states

Stéphane Demri

LSV, ENS Cachan, CNRS, INRIA Saclay

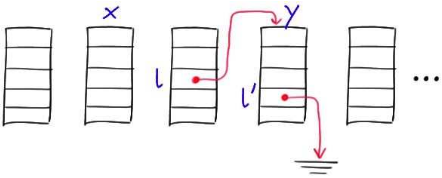
Joint work with Rémi Brochenin and Etienne Lozes

November 14th, 2008 — Séminaire MeFoSyLoMa

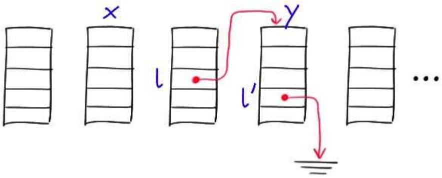
Pointer programs

- Pointer: reference to a memory cell
  (non fixed memory address).
- Dynamic memory allocation/deallocation.
- Examples of instructions:
  - $\mathtt{x} := \mathtt{y}$: assign the value $\mathtt{y}$ to the variable $\mathtt{x}$,
  - $\mathtt{x} := \mathtt{y} \to l$: read the l-field of the cell pointed to by $\mathtt{y}$ into $\mathtt{x}$,
  - $\mathtt{y} \to l := \mathtt{x}$: write $\mathtt{x}$ to the l-field of the cell pointed to by $\mathtt{y}$,
  - `free x`: deallocate the cell pointer to by x,
  - $\mathtt{x} := \mathtt{malloc}(i)$: allocate i memory cells and assign its address to $\mathtt{x}$.
- Simple safety properties of pointer programs are undecidable
  ("there is no `null` dereference").

Memory states



Memory states



- Set of variables $\mathtt{Var}$.
- Set of labels $\mathtt{Lab}$.
- Set of values $\mathtt{Val} = \mathbb{N} \uplus \{nil\}$.

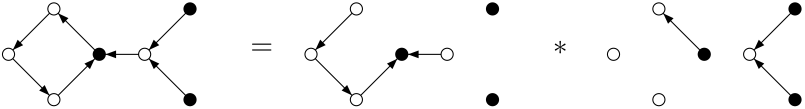
- Set of stores: $\mathcal{S} \stackrel{\text{def}}{\equiv} \mathtt{Var} \to \mathtt{Val}$.
- Set of heaps:
  $\mathcal{H} \stackrel{\text{def}}{\equiv} \mathbb{N} \rightharpoonup_{fin} (\mathtt{Lab} \rightharpoonup_{fin+} \mathtt{Val})$.
- Memory state: (s, h).

Disjoint heaps

- h_1 and h_2 are disjoint whenever $\mathtt{dom}(h_1) \cap \mathtt{dom}(h_2) = \emptyset$.
  Notation: $h_1 \perp h_2$.
- Disjointness does not concern records.
- Disjoint union $h_1 * h_2$ whenever $h_1 \perp h_2$.

Disjoint heaps

- h_1 and h_2 are disjoint whenever $\mathtt{dom}(h_1) \cap \mathtt{dom}(h_2) = \emptyset$.
  Notation: $h_1 \perp h_2$.
- Disjointness does not concern records.
- Disjoint union $h_1 * h_2$ whenever $h_1 \perp h_2$.
- Disjoint heaps (with a unique label):



Analysis of pointer programs

- Memory leak: a memory cell can no longer be reached.
- Null-pointer dereferencing.
- Alias analysis: checking whether memory cells are shared.
- Shape analysis: checking the structure of the heap.
- Functional properties: compare input and output heaps, data properties.

$\Rightarrow$ Verification of program with pointers requires *fine-tuned* specification languages to speak about memory states and their evolution.

Reasoning about pointer programs

- Separation logic [Reynolds, LICS 02].
- Pointer assertion logic (PAL) [Jensen et al. 97]. Monadic 2nd logic whose the universe of discourse contains records, pointers and booleans (non-elementary complexity)
- TVLA [Lev-Ami & Sagiv, SAS 00]: abstract interpretation technique with Kleene's logic (op. semantics in FOL + TC)
- Alias logic [Bozga & Iosif & Lakhnech, SAS 04].
- Logic of Reachable Patterns [Yorsh et al., FOSSACS 06].
- Evolution Logic [Yahav et al., ESOP 03]: to specify temporal properties of programs with dynamically evolving heaps.

Model checking

- Navigation Temporal Logic
  [Distefano & Katoen & Rensink, FSTTCS 04].
- Bounded model-checking
  [Charatonik & Georgieva & Maier, CSL 05].
  Decidability for a fragment of FOL with Datalog programs.
- Model-checking pointer systems
  [Bardin & Finkel & Nowak, AVIS 04; Bardin, PhD 05].
- Regular model-checking [Bouajjani et al., TACAS 05].
- Translation into counter automata
  [Bouajjani et al, CAV 06; Sangnier, PhD 08].

Our motivations

- To design temporal languages to specify the behaviors of pointer programs.
- To combine an assertion language from separation logic with linear-time/branching-time temporal logics.
- To evaluate the borders for decidability.
- To admit effective procedure with “reasonable” computational complexity for precise analysis.
- Automata-based proof technique with symbolic memory states.

Separation logic

- Introduced by Reynolds, Pym and O'Hearn.
- Reasoning about the heap with a strong form of locality built-in.
- $\mathcal{A} * \mathcal{B}$ is true whenever the heap can be divided into two disjoint parts, one satisfies $\mathcal{A}$, the other one $\mathcal{B}$.
- $\mathcal{A} \mathrel{-\!\!*} \mathcal{B}$ is true whenever $\mathcal{A}$ is true for a (fresh) disjoint heap, $\mathcal{B}$ is true for the combined heap.
- Hoare-style proof system for local reasoning about pointer programs, e.g. frame rule:

$$\frac{\{\mathcal{A}\}\ \texttt{PROG}\ \{\mathcal{B}\}}{\{\mathcal{A} * \mathcal{B}'\}\ \texttt{PROG}\ \{\mathcal{B} * \mathcal{B}'\}}$$

Hoare triples

- Hoare triple: $\{\mathcal{A}\}$ `PROG` $\{\mathcal{B}\}$.
- Total correctness: if we start in a state where $\mathcal{A}$ holds true and execute `PROG`, the program `PROG` will terminate in a state satisfying $\mathcal{B}$.
- Hoare logic uses Hoare triples to reason about program correctness.
- Rule of constancy:

$$\frac{\{\mathcal{A}\}\ \texttt{PROG}\ \{\mathcal{B}\}}{\{\mathcal{A} \wedge \mathcal{B}'\}\ \texttt{PROG}\ \{\mathcal{B} \wedge \mathcal{B}'\}}$$

  where no variable free in $\mathcal{B}'$ is modified by `PROG`.

When separation logic enters into the play

- Unsoundness of the rule of constancy in separation logic:

$$\frac{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z})\}\ [\mathtt{x}] := 4\ \{\mathtt{x} \mapsto 4\}}{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z}) \wedge \mathtt{y} \mapsto 3\}\ [\mathtt{x}] := 4\ \{\mathtt{x} \mapsto 4 \wedge \mathtt{y} \mapsto 3\}}$$

(when $\mathtt{x} = \mathtt{y}$)

- Reparation with frame rule:

$$\frac{\{\mathcal{A}\}\ \mathtt{PROG}\ \{\mathcal{B}\}}{\{\mathcal{A} * \mathcal{B}'\}\ \mathtt{PROG}\ \{\mathcal{B} * \mathcal{B}'\}}$$

where no variable free in $\mathcal{B}'$ is modified by $\mathtt{PROG}$.

Standard inference rules for mutation

- Local form (MUL)

$$\overline{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z})\}\ [\mathtt{x}] := \mathtt{y}\ \{\mathtt{x} \mapsto \mathtt{y}\}}$$

- Global form (MUG)

$$\overline{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z}) * \phi\}\ [\mathtt{x}] := \mathtt{y}\ \{\mathtt{x} \mapsto \mathtt{y} * \mathcal{A}\}}$$

- Backward-reasoning form (MUBR)

$$\overline{\{(\exists \mathtt{z}.\ \mathtt{x} \mapsto \mathtt{z}) * ((\mathtt{x} \mapsto \mathtt{y}) \mathrel{-\!\!*} \mathcal{A})\}\ [\mathtt{x}] := \mathtt{y}\ \{\mathcal{A}\}}$$

Separation Logics (SL)

- Expressions

$$e ::= \mathtt{x} \mid \mathtt{null}$$

- Atomic formulae

$$\pi ::= \; e = e' \mid \; \mathtt{x} + i \stackrel{l}{\hookrightarrow} e$$

- Standard $e \hookrightarrow e', e"$ can be encoded with $e \stackrel{1}{\hookrightarrow} e' \wedge e \stackrel{2}{\hookrightarrow} e"$.

- $i = 0$ for no arithmetics on pointers.

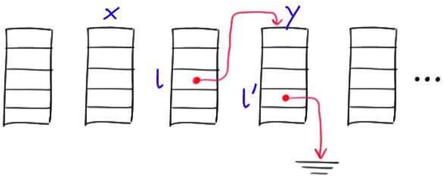
- State formulae

$$\mathcal{A} ::= \mathtt{emp} \mid \pi \mid \mathcal{A} \wedge \mathcal{B} \mid \neg\mathcal{A} \mid \mathcal{A} * \mathcal{B} \mid \mathcal{A} {-\!\!*} \mathcal{B}$$

Semantics

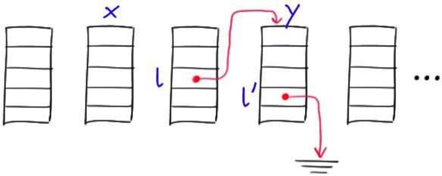
- $(s, h) \models_{\mathrm{SL}} \mathtt{emp}$ iff $\mathtt{dom}(h) = \emptyset$.
- $(s, h) \models_{\mathrm{SL}} e = e'$ iff $[\![\ e\ ]\!]_s = [\![\ e'\ ]\!]_s$, with $[\![\ \mathtt{x}\ ]\!]_s = s(\mathtt{x})$ and $[\![\ \mathtt{null}\ ]\!]_s = nil$.
- $(s, h) \models_{\mathrm{SL}} \mathtt{x} + i \overset{l}{\hookrightarrow} e'$ iff $[\![\ \mathtt{x}\ ]\!]_s \in \mathbb{N}$ and $[\![\ \mathtt{x}\ ]\!] + i \in \mathtt{dom}(h)$ and $h(s(\mathtt{x}) + i)(l) = [\![\ e'\ ]\!]_s$.
- $(s, h) \models_{\mathrm{SL}} \mathcal{A}_1 * \mathcal{A}_2$ iff $\exists\ h_1, h_2$ such that $h = h_1 * h_2$, $(s, h_1) \models_{\mathrm{SL}} \mathcal{A}_1$ and $(s, h_2) \models_{\mathrm{SL}} \mathcal{A}_2$.
- $(s, h) \models_{\mathrm{SL}} \mathcal{A}_1 \mathrel{-\!\!*} \mathcal{A}_2$ iff for all h', if $h \perp h'$ and $(s, h') \models_{\mathrm{SL}} \mathcal{A}_1$ then $(s, h * h') \models_{\mathrm{SL}} \mathcal{A}_2$.
- $+$ clauses for Boolean operators.

Memory states with arithmetic and records



$$\mathtt{x}{+}1 \overset{l}{\hookrightarrow} \mathtt{y}$$
$$\mathtt{y} \overset{l'}{\hookrightarrow} \mathtt{null}$$

Memory states with arithmetic and records



$$\mathtt{x}{+}1 \overset{l}{\hookrightarrow} \mathtt{y} \qquad h(s(\mathtt{x})+1)(l) = s(\mathtt{y})$$

$$\mathtt{y} \overset{l'}{\hookrightarrow} \mathtt{null} \qquad h(s(\mathtt{y}))(l') = nil$$

Simple properties on memory states

- The memory heap has at least two cells ($\mathtt{size} \geq 2$):

$$\neg\mathtt{emp} * \neg\mathtt{emp}$$

- The memory heap has exactly one cell at address $\mathtt{x}$ ($\mathtt{x} \overset{l}{\mapsto} e$):

$$\mathtt{x} \overset{l}{\hookrightarrow} e \wedge \neg(\mathtt{size} \geq 2)$$

- The variable $\mathtt{x}$ is allocated in the heap ($\mathtt{alloc}(\mathtt{x})$):

$$(\mathtt{x} \overset{l}{\hookrightarrow} \mathtt{null}) -\!\!* \perp$$

On the complexity of SL

- Model-checking, satisfiability and validity for SL are PSPACE-complete problems.
- PSPACE-hardness is from [Calcagno & Yang & O'Hearn, FSTTCS 01].
- PSPACE upper bound is obtained thanks to a "small memory state property".
- PSPACE upper bound of SL without arithmetics can be obtained by translation into a "separation-free" version. [Lozes, SPACE 04].
- $\mathrm{SL} + \exists$ is undecidable [C. & Y. & O'H., FSTTCS 01]. even with a unique label [BDL'08].

Small store property

- Standard property: $\mathcal{A}$ is satisfiable iff there is a store s such that $(s, \emptyset) \models_{\mathrm{SL}} \neg(\mathcal{A} \mathrel{-\!\!*} \bot)$.
- Refinement: $\mathcal{A}$ is satisfiable iff there is a store s such that
  - $(s, \emptyset) \models_{\mathrm{SL}} \neg(\mathcal{A} \mathrel{-\!\!*} \bot)$,
  - for each variable $\mathtt{x} \in Y$, $s(\mathtt{x}) \leq (|Y| + 1) \times max\ \epsilon$,

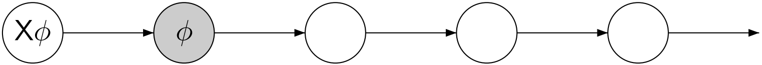
  where
  - Y is the set of variables occuring in $\mathcal{A}$,
  - ϵ is the set of indices i such that $\mathtt{x} + i$ occurs in $\mathcal{A}$ for some variable $\mathtt{x}$.

Temporal Separation Logic

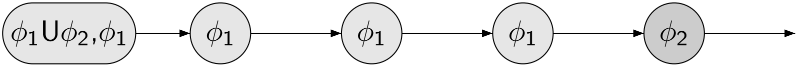
- To combine spatial properties and temporal properties
  - What are the modes of combination?
    See e.g. multidimensional logics in [Gabbay et al., Book 03].
  - Which problems are decidable?
    LTL with zero tests and incrementation is undecidable.
  - How the memory states are updated?
    constant heap, programs without destructive update, etc.
- To add recursion in SL.
- To extend the automata-based approach for model-checking?
  [Vardi & Wolper, IC 94].
- LTL over concrete domains
  See e.g., [Esparza, ICALP 94; Demri & D'Souza, IC 07].

LTL operators in a nutshell

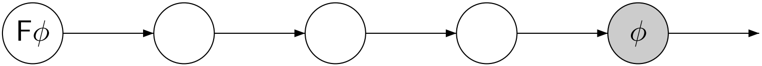
$X\phi$: next-time ϕ



$\phi_1 U \phi_2$: ϕ_1 until ϕ_2



$F\phi$: sometimes ϕ



About plain LTL

- Formulae: $\phi ::= \mathsf{p} \mid \neg\phi \mid \phi \wedge \psi \mid \phi \mathsf{U} \psi \mid \mathsf{X}\phi$.
- Models: σ: $\mathbb{N} \rightarrow \mathcal{P}(\mathrm{PROP})$ and $\sigma, i \models p$ iff $p \in \sigma(i)$.
- $\mathrm{L}(\phi) = \{\sigma \in (\mathcal{P}(\mathrm{PROP}))^\omega : \sigma, 0 \models \phi\}$.
- $\phi \rightsquigarrow$ Büchi automaton $\mathbb{A}_\phi$ such that $\mathrm{L}(\phi) = \mathrm{L}(\mathbb{A}_\phi)$.
  [Vardi & Wolper, IC 94].
- $|\mathbb{A}_\phi|$ is in $2^{\mathcal{O}(|\phi|)}$.
- Model-checking and satisfiability are PSPACE-complete.
  [Sistla & Clarke, JACM 85].

The logic $\mathrm{LTL}^{\mathrm{mem}}$

- Syntax

$$
\begin{array}{llr}
e ::= & \mathtt{x} \mid \mathtt{null} \mid \mathsf{X}e & \text{(expressions)} \\
\pi ::= & e = e' \mid e + i \stackrel{l}{\hookrightarrow} e & \text{(atomic formulae)} \\
\mathcal{A} ::= & \pi \mid \mathcal{A} \wedge \mathcal{B} \mid \neg \mathcal{A} & \text{(classical fragment)} \\
 & \mid \mathcal{A} * \mathcal{B} \mid \mathcal{A} \mathrel{-\!\!*} \mathcal{B} \mid \mathtt{emp} & \text{(spatial fragment)} \\
\phi ::= & \mathcal{A} \mid \mathsf{X}\phi \mid \phi \mathsf{U} \phi' \mid \phi \wedge \phi' \mid \neg \phi & \text{(temporal formulae)}
\end{array}
$$

- Examples

$$\mathsf{G}\ (\mathtt{alloc}(\mathtt{x}) \Rightarrow \mathsf{F}\ \mathtt{alloc}(\mathtt{y}))$$

$$\mathsf{GF}(\mathtt{size} \geq 2) \qquad (\mathsf{X}\mathtt{x} = \mathtt{x})\mathsf{U}(\mathtt{y} \stackrel{l}{\hookrightarrow} \mathtt{z})$$

Semantics

Models: elements of $(\mathcal{S} \times \mathcal{H})^\omega$ of the form $\rho = (s_i, h_i)_{i \geq 0}$.

$$
\begin{array}{lll}
\rho, t \models & e = e' & \text{iff } [\![\ e\ ]\!]_{\rho,t} = [\![\ e'\ ]\!]_{\rho,t} \quad \textit{with } [\![\ Xe\ ]\!]_{\rho,t} = [\![\ e\ ]\!]_{\rho,t+1} \\
\rho, t \models & e + i \stackrel{l}{\hookrightarrow} e' & \text{iff } h_t([\![\ e\ ]\!]_{\rho,t} + i) = [\![\ e'\ ]\!]_{\rho,t} \\
\rho, t \models & \mathcal{A}_1 * \mathcal{A}_2 & \text{iff } \exists\ h_1, h_2 \text{ s.t. } h_t = h_1 * h_2, \\
 & & \quad \rho[h_t \leftarrow h_1], t \models \mathcal{A}_1, \\
 & & \quad \text{and } \rho[h_t \leftarrow h_2], t \models \mathcal{A}_2. \\
\rho, t \models & \mathcal{A}_1 \mathrel{-\!\!*} \mathcal{A}_2 & \text{iff } \forall h',\ \text{if } h_t \perp h' \text{ and } \rho[h_t \leftarrow h'], t \models \mathcal{A}_1 \\
 & & \quad \text{then } \rho[h_t \leftarrow h * h'], t \models \mathcal{A}_2. \\
\rho, t \models & \mathsf{X}\phi & \text{iff } \rho, t+1 \models \phi. \\
\rho, t \models & \phi \mathsf{U} \phi' & \text{iff } \exists t' \geq t \text{ such that } \rho, t' \models \phi', \\
 & & \quad \text{and } \forall t'',\ t \leq t'' < t',\ \rho, t'' \models \phi.
\end{array}
$$

Satisfiability problems

- Satisfiability problem $\mathrm{SAT}(\mathrm{Frag})$ with underlying fragment $\mathrm{Frag} \subseteq \mathrm{SL}$.
- Problem $\mathrm{SAT}^{ct}(\mathrm{Frag})$ with constant heap
  $\rightarrow$ temporal language allows us to explore the heap.
- Problem $\mathrm{SAT}_{init}(\mathrm{Frag})$ with a fixed initial heap.

A class of programs manipulating pointers

- Set of instructions

$$
\begin{aligned}
\texttt{instr} ::= \quad & \texttt{x} := \texttt{y} \mid \texttt{skip} \\
& \mid \texttt{x} := \texttt{y} \rightarrow l \mid \texttt{x} \rightarrow l := \texttt{y} \\
& \mid \texttt{x} := \texttt{cons}(l_1 : x_1, .., l_k : x_k) \mid \texttt{free}\ \texttt{x}, l \\
& \mid \texttt{x} := \texttt{y}[i] \mid \texttt{x}[i] := \texttt{y} \\
& \mid \texttt{x} = \texttt{malloc}(i) \mid \texttt{free}\ \texttt{x}, i
\end{aligned}
$$

- Programs are finite-state automata with transitions labelled by instructions and equality tests.

- A program without destructive update admits runs with constant heap.

Model-checking problems

- $\mathrm{MC}(\mathrm{Frag})$: given ϕ in $\mathrm{LTL}^{\mathrm{mem}}$ with state formulae built over Frag and a program `PROG` of the associated fragment, is there an infinite computation ρ of `PROG` such that $\rho, 0 \models \phi$?

- $\mathrm{MC}^{ct}_{init}(\mathrm{Frag})$: idem with fixed initial memory state and no destructive update.

Fragments with decidable temporal reasoning

- SL fragments:

Classical fragment (CL)

$$\mathcal{A} ::= \quad e = e' \mid \mathtt{x} + i \overset{l}{\hookrightarrow} e \mid \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A}$$

Record fragment (RF)

$$\mathcal{A} ::= \quad e = e' \mid \mathtt{x} \overset{l}{\hookrightarrow} e \mid \mathcal{A} * \mathcal{A} \mid \mathcal{A} \mathrel{-\!\!*} \mathcal{A} \mid \mathtt{emp} \mid \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A}$$

- Theorem: The satisfiability problems for $\mathrm{LTL}^{\mathrm{mem}}$(CL) and $\mathrm{LTL}^{\mathrm{mem}}$(RF) are PSPACE-complete.

Bounding the syntactic resources

- Test formulae

$$e ::= \langle \mathtt{x}, u \rangle \mid \mathtt{null} \qquad f ::= e + i$$
$$\psi ::= f \stackrel{l}{\hookrightarrow} e \mid \mathtt{alloc}(f) \mid e = e' \mid \mathtt{size} \geq k$$

  - $u, i, k \in \mathbb{N}$,
  - u encoded in unary since $\langle \mathtt{x}, u \rangle \approx \mathsf{X}^u\mathtt{x}$,
  - $\mathtt{x}$ is a variable and l is a label.

- Measure μ restricts the test formulae

$$\mu = (m, \epsilon, w, X, Y) \in \mathbb{N} \times \mathcal{P}_f(\mathbb{N}) \times \mathbb{N} \times \mathcal{P}_f(\mathtt{Lab}) \times \mathcal{P}_f(\mathtt{Var})$$

- $\mathcal{T}_\mu$: set of test formulae restricted to the resources from the measure.

Symbolic models and abstraction

- Symbolic model: $\sigma \ : \ \mathbb{N} \rightarrow \mathcal{P}(\mathcal{T}_\mu)$.
- Abstraction: $\rho \in (\mathcal{S} \times \mathcal{H})^\omega \mapsto \mathit{Abs}_\mu(\rho) \in \mathcal{P}(\mathcal{T}_\mu)^\omega$.

$$\mathit{Abs}_\mu(\rho)(i) \stackrel{\text{def}}{=} \{\mathcal{A} \in \mathcal{T}_\mu \ : \ \rho, i \models \mathcal{A}\}.$$

- See also resource graphs in [Galmiche & Mery, JLC'08].
- Symbolic satisfaction relation: $\sigma, i \models_\mu \phi$ defined by induction on ϕ with the base case: $\sigma, i \models_\mu \mathcal{A} \ \stackrel{\text{def}}{\Leftrightarrow}$

$$\models_{\mathrm{SL}} \Big(\bigwedge_{\mathcal{A}' \in \sigma(i)} \mathcal{A}' \ \wedge \bigwedge_{\mathcal{A}' \in (\mathcal{T}_\mu \setminus \sigma(i))} \neg \mathcal{A}' \Big) \ \Rightarrow \ \mathcal{A}$$

Checking satisfiability with symbolic models

- **Lemma:** ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ is satisfiable iff there is a symbolic model $\sigma : \mathbb{N} \to \mathcal{P}(\mathcal{T}_{\mu_\phi})$ such that
  - σ symbolically satisfies ϕ ($\sigma, 0 \models_{\mu_\phi} \phi$)
  - there is a model ρ of $\mathrm{LTL}^{\mathrm{mem}}$ such that $Abs_\mu(\rho) = \sigma$.

- For instance, $\{\mathsf{X}\mathtt{x} = \mathsf{X}^2\mathtt{x}, \ldots\}, \{\mathtt{x} \neq \mathsf{X}\mathtt{x}, \ldots\} \ldots$ has no concrete models.

The generalized Büchi automaton $\mathbb{A}^{\mu}_{\phi}$

- Q is the set of atoms of ϕ (sets of subformulae).
- $I = \{X \in Q : \phi \in X\}$.
- $\Sigma = \mathcal{P}(\mathcal{T}_{\mu})$.
- $X \xrightarrow{a} Y$ iff
  1. for every atomic formula $\mathcal{A}$ of X, $\models_{\mathrm{SL}} \mathcal{A}_a \Rightarrow \mathcal{A}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u \rangle]$.
  2. for every $\mathsf{X}\phi' \in cl(\phi)$, $\mathsf{X}\phi' \in X$ iff $\phi' \in Y$.
- Let $\{\phi_1 \mathsf{U} \phi'_1, \ldots, \phi_n \mathsf{U} \phi'_n\}$ be the set of until formulae in $cl(\phi)$. We pose $\mathcal{F} = \{F_1, \ldots, F_n\}$ where $F_i = \{X \in Q : \phi_i \mathsf{U} \phi'_i \notin X \text{ or } \phi'_i \in X\}$ for $i \in \{1, \ldots, n\}$.

- Lemma: Let ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ and $\mu \geq \mu_{\phi}$. Then, $\mathrm{L}(\mathbb{A}^{\mu}_{\phi})$ is the set of symbolic models satisfying ϕ.

The automaton $\mathbb{A}^{\mu}_{sat}$ for consistency

- $\Sigma = \mathcal{P}(\mathcal{T}_\mu)$, $Q = I = F = \Sigma$,
- $a \xrightarrow{a'} a''$ iff:
    1. $\mathcal{A}_a, \mathcal{A}_{a''}$ are satisfiable, and $a = a'$,
    2. for every formula $\langle \mathtt{x}, u \rangle = \langle \mathtt{x}', u' \rangle \in \mathcal{T}_\mu$ with $u, u' \geq 1$, $\langle \mathtt{x}, u \rangle = \langle \mathtt{x}', u' \rangle \in a$ iff $\langle \mathtt{x}, u-1 \rangle = \langle \mathtt{x}', u'-1 \rangle \in a''$.
- Lemma: Let ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ and $\mu = \mu_\phi$. Then $\mathrm{L}(\mathbb{A}^{\mu}_{sat})$ is the set of symbolic models being the abstraction of some concrete model.

Other decidable satisfiability problems

- $\mathrm{SAT}^{ct}_{init}(\mathrm{Frag})$: satisfiability problem of the fragment Frag with fixed initial memory state and constant heap models.

- Theorem:
  - $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$ is PSPACE-complete.
    Proof by reduction to $\mathrm{SAT}(\mathrm{RF})$ by internalizing the initial memory state and the fact that the heap is constant.
  - $\mathrm{SAT}^{ct}_{init}(\mathrm{CL})$ is PSPACE-complete.
    Similar internalization.
  - $\mathrm{SAT}^{ct}_{init}(\mathrm{SL} \setminus -\!\!*)$ is PSPACE-complete.
    Proof by reduction to $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$ in order to eliminate the arithmetic expressions.

Other decidable problems

- $\mathrm{MC}^{ct}_{init}(\mathrm{RF})$ is PSPACE-complete.
  Proof by reduction into $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$.
- $\mathrm{MC}^{ct}_{init}(\mathrm{SL})$ is PSPACE-complete.
  Proof by reduction into LTL model-checking.
- Replacing X and U by a finite set of MSO definable preserves the PSPACE upper bound.

Satisfiability problems

- Theorem: $\mathrm{SAT}^{?}_{?}(\mathrm{SL})$ and $\mathrm{SAT}(\mathrm{SL} \setminus \mathrel{-\!\!*})$ are Σ^1_1-complete.
- Proof by reducing the recurrence problem for ND Minsky machines [Alur & Henzinger, JACM 94].
- Incrementation is encoded thanks to

$$(\mathsf{X}\mathtt{x} \hookrightarrow \mathtt{y} \ \wedge \ \mathtt{x}+1 \hookrightarrow \mathtt{y}) \ \ \wedge \ \ \neg\,(\mathsf{X}\mathtt{x} \hookrightarrow \mathtt{y} \ * \ \mathtt{x}+1 \hookrightarrow \mathtt{y})$$

An undecidable model-checking problem

- List fragment LF: RF with a unique label.
- Theorem: $\mathrm{MC}^{ct}(\mathrm{LF})$ is Σ^0_1-complete.
- Reduction from the halting problem for Minsky machines.
- Maximal value of counters:

$$_{\mathtt{z}}\square \xrightarrow{next} \square \xrightarrow{next} \cdots \square \xrightarrow{next} \square \xrightarrow{next} nil$$

- The length of the list starting at $\mathtt{x}_i$ encodes the value of the counter C_i.
- Preliminary verification to check that $\mathtt{z}$ points to a list.
- Decrementing C_i is simulated by $\mathtt{x}_i := \mathtt{x}_i \rightarrow next$.

Summary of main complexity results

	MC	MC^{ct}	MC^{ct}_{init}	SAT	SAT^{ct}	SAT^{ct}_{init}
LF	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c.	PSPACE-c.	Σ^0_1-c.	PSPACE-c.
CL and RF	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c.	PSPACE-c.	Σ^0_1-c.	PSPACE-c.
$\mathrm{SL}\backslash\{-\!\!*\}$	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c
SL	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c	Σ^1_1-c.	Σ^1_1-c.	Σ^1_1-c.

Conclusion and perspectives

- Introduction of a logic mixing temporal operators and assertions from separation logic.
- Characterization of the complexity of model-checking and satisfiability problems for fragments and under different hypotheses.
- Some open problems:
  - Which classes of constraints on successive heaps restore decidability?
  - How to add recursion to separation logic while preserving decidability?

Some bibliographical references

- Separation logic and verification
  [Reynolds, LICS 02]
- Complexity results on separation logic
  [Calcagno & O'Hearn & Yang, FSTTCS 01]
- Propositional separation logic expressiveness
  [Lozes, SPACE 04]
- Tableaux and resource graphs for separation logic
  [Galmiche & Mery, JLC 08]
- LTL over concrete domains
  [Demri & D'Souza, IC 07; Gascon, PhD 07]