C05 – Separation Logic & SAT solvers

Program Verification

FMI · Denisa Diaconescu · Spring 2021

Separation Logic

Adding the heap

We extend our programming language with:

- Heap reads: `x := [`$\mathbb{E}$`]` *(dereferencing)*
- Heap writes: `[`$\mathbb{E}_1$`] := `$\mathbb{E}_2$ *(update heap)*
- Heap allocation: `x := cons(`$\mathbb{E}_1$`,...`$\mathbb{E}_n$`)`
- Heap deallocation: `dispose` $\mathbb{E}$

The state is now represented by a pair of type $\mathit{Store} \times \mathit{Heap}$, denoted (σ, h), where

$$\sigma \in \mathit{Store}, \text{ where } \mathit{Store} \triangleq \mathit{Var} \to \mathit{Val}$$

$$h \in \mathit{Heap}, \text{ where } \mathit{Heap} \triangleq \mathit{Loc} \to \mathit{Val}$$

where $\mathit{Loc} \subseteq \mathit{Val}$.

Note that we consider $\mathit{dom}(h)$ to always be finite. By this, we ensure that `cons` commands will never fail.

Evaluating expressions in the store of a state

Strictly speaking, the store gives values to variables only.

But we need a way to say "value of an expression in a store" so we will abuse notation and use $\sigma(\mathbb{E})$ for this as below:

- $\sigma(n) = n$ where n is a number is just its usual value
- $\sigma(x + n) = \sigma(x) + \sigma(n)$ where n is a number and x is a variable

Extra connectives in separation logic

$\texttt{emp}$	empty heap
$\mathbb{E}_1 \mapsto \mathbb{E}_2$	points to
$P \ast Q$	separating conjunction

Semantics of separation logic

$\sigma \triangleq \mathit{Var} \to \mathit{Val}$

$h \triangleq \mathit{Loc} \to \mathit{Val}$

$(\sigma, h) \models \texttt{emp}$ if $\mathit{dom}(h) = \emptyset$

- emp is an atomic formula for checking if the heap is empty
- a state (σ, h) makes the formula emp true if the heap is empty

Semantics of separation logic

$\sigma \triangleq \mathit{Var} \to \mathit{Val}$

$h \triangleq \mathit{Loc} \to \mathit{Val}$

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $\mathit{dom}(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

- a state (σ, h) makes the formula $\mathbb{E}_1 \mapsto \mathbb{E}_2$ true if the heap is a singleton and maps the location $\sigma(\mathbb{E}_1)$ to the value $\sigma(\mathbb{E}_2)$
- $\sigma(\mathbb{E})$ is the value of an expression in a store as explained before

Semantics of separation logic

$\sigma \triangleq \mathit{Var} \to \mathit{Val}$

$h \triangleq \mathit{Loc} \to \mathit{Val}$

$(\sigma, h) \models P * Q$ if h can be partitioned into two disjoint heaps h_1 and h_2,

and $(\sigma, h_1) \models P$ and $(\sigma, h_2) \models Q$

Note that two heaps are disjoint if the intersection of their domains is empty.

Semantics of separation logic

$\sigma \triangleq Var \rightarrow Val$

$h \triangleq Loc \rightarrow Val$

$(\sigma, h) \models P * Q$ if h can be partitioned into two disjoint heaps h_1 and h_2,

and $(\sigma, h_1) \models P$ and $(\sigma, h_2) \models Q$

Note that two heaps are disjoint if the intersection of their domains is empty.

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n disjoint heaps

$h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Semantics of separation logic

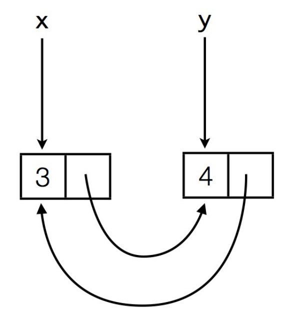
Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Semantics of separation logic

Example

σ

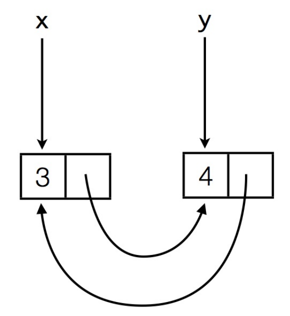
x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

Satisfies the statement:

$(x \mapsto 3) * (x + 1 \mapsto y) * (y \mapsto 4) * (y + 1 \mapsto x)$



Semantics of separation logic

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n distinct heaps $h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

We want to show that

$$(\sigma, h) \models (\mathtt{x} \mapsto 3) * (\mathtt{x} + 1 \mapsto \mathtt{y}) * (\mathtt{y} \mapsto 4) * (\mathtt{y} + 1 \mapsto \mathtt{x})$$

Semantics of separation logic

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n distinct heaps $h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

We want to show that

$$(\sigma, h) \models (\mathtt{x} \mapsto 3) * (\mathtt{x} + 1 \mapsto \mathtt{y}) * (\mathtt{y} \mapsto 4) * (\mathtt{y} + 1 \mapsto \mathtt{x})$$

We can partition h into 4 distinct heaps:

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

Semantics of separation logic

$(\sigma, h) \models P_1 * P_2 * \ldots * P_n$ if h can be partitioned into n distinct heaps $h_1, h_2, \ldots, h_n$ and $(\sigma, h_i) \models P_i$ for any $i \in \{1, \ldots, n\}$

Example

σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

We want to show that

$$(\sigma, h) \models (\mathtt{x} \mapsto 3) * (\mathtt{x} + 1 \mapsto \mathtt{y}) * (\mathtt{y} \mapsto 4) * (\mathtt{y} + 1 \mapsto \mathtt{x})$$

We can partition h into 4 distinct heaps:

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

We must show that

$(\sigma, h_1) \models \mathtt{x} \mapsto 3$

$(\sigma, h_2) \models \mathtt{x} + 1 \mapsto \mathtt{y}$

$(\sigma, h_3) \models \mathtt{y} \mapsto 4$

$(\sigma, h_4) \models \mathtt{y} + 1 \mapsto \mathtt{x}$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = \text{0xAB} = \sigma(\mathtt{x})$
- $h_1(\text{0xAB}) = 3$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = \text{0xAB} = \sigma(\mathtt{x})$
- $h_1(\text{0xAB}) = 3$

$(\sigma, h_2) \models x + 1 \mapsto y$

- $dom(h_2) = \text{0xAC} = \sigma(\mathtt{x} + 1)$
- $h_2(\text{0xAC}) = \text{0xDD} = \sigma(\mathtt{y})$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = \text{0xAB} = \sigma(\mathtt{x})$
- $h_1(\text{0xAB}) = 3$

$(\sigma, h_2) \models x + 1 \mapsto y$

- $dom(h_2) = \text{0xAC} = \sigma(\mathtt{x} + 1)$
- $h_2(\text{0xAC}) = \text{0xDD} = \sigma(\mathtt{y})$

$(\sigma, h_3) \models \mathtt{y} \mapsto 4$

- $dom(h_3) = \text{0xDD} = \sigma(\mathtt{y})$
- $h_3(\text{0xDD}) = 4$

Semantics of separation logic

$(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$

Example

σ

x	0xAB
y	0xDD

h_1

0xAB	3

h_2

0xAC	0xDD

h_3

0xDD	4

h_4

0xDE	0xAB

$(\sigma, h_1) \models x \mapsto 3$

- $dom(h_1) = \text{0xAB} = \sigma(\mathtt{x})$
- $h_1(\text{0xAB}) = 3$

$(\sigma, h_2) \models x + 1 \mapsto y$

- $dom(h_2) = \text{0xAC} = \sigma(\mathtt{x} + 1)$
- $h_2(\text{0xAC}) = \text{0xDD} = \sigma(\mathtt{y})$

$(\sigma, h_3) \models \mathtt{y} \mapsto 4$

- $dom(h_3) = \text{0xDD} = \sigma(\mathtt{y})$
- $h_3(\text{0xDD}) = 4$

$(\sigma, h_4) \models \mathtt{y} + 1 \mapsto \mathtt{x}$

- $dom(h_4) = \text{0xDE} = \sigma(\mathtt{y} + 1)$
- $h_4(\text{0xDE}) = \text{0xAB} = \sigma(\mathtt{x})$

Semantics of separation logic

Example

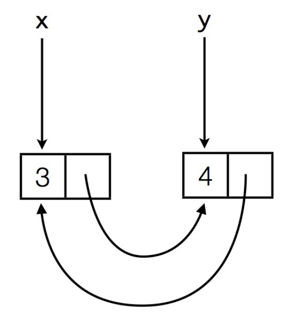
σ

x	0xAB
y	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB

Doest not satisfy the statement $\mathtt{x} \mapsto 3$



Semantics of separation logic

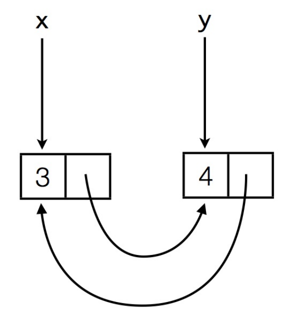
Example

σ

$\mathtt{x}$	0xAB
$\mathtt{y}$	0xDD

h

0xAB	3
0xAC	0xDD
0xDD	4
0xDE	0xAB



Doest not satisfy the statement $\mathtt{x} \mapsto 3$

- $(\sigma, h) \models \mathbb{E}_1 \mapsto \mathbb{E}_2$ if $dom(h) = \{\sigma(\mathbb{E}_1)\}$ and $h(\sigma(\mathbb{E}_1)) = \sigma(\mathbb{E}_2)$
- $dom(h) = \{0xAB, 0xAC, 0xDD, 0xDE\}$
- $\sigma(\mathtt{x}) = 0xAB$
- $h(\sigma(\mathtt{x})) = 3$

Store assignment axiom for separation logic

Hoare axiom: $\{Q[\mathtt{x}/\mathbb{E}]\}$ `x := ` $\mathbb{E}$ $\{Q\}$

Floyd axiom: $\{\mathtt{x} = v\}$ `x := ` $\mathbb{E}$ $\{\mathtt{x} = \mathbb{E}[\mathtt{x}/v])\}$

where v is an auxiliary variable which does not occur in $\mathbb{E}$.

Store assignment axiom for separation logic

Hoare axiom: $\{Q[\mathtt{x}/\mathbb{E}]\}\ \mathtt{x\ :=\ }\mathbb{E}\ \{Q\}$

Floyd axiom: $\{\mathtt{x} = v\}\ \mathtt{x\ :=\ }\mathbb{E}\ \{\mathtt{x} = \mathbb{E}[\mathtt{x}/v])\}$

where v is an auxiliary variable which does not occur in $\mathbb{E}$.

Store assignment axiom for Separation logic:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ }\mathbb{E}\ \{\mathtt{x} = \mathbb{E}[\mathtt{x}/v] \wedge \mathtt{emp}\}$$

where v is an auxiliary variable which does not occur in $\mathbb{E}$

New:

- atomic formula `emp` to say that the "heap is empty"
- we want to track the smallest amount of heap information

Store assignment axiom for separation logic

Store assignment axiom for Separation logic:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ }\mathbb{E}\ \{\mathtt{x} = \mathbb{E}[\mathtt{x}/v] \wedge \mathtt{emp}\}$$

where v is an auxiliary variable which does not occur in $\mathbb{E}$

Example

$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ 1}\ \{\mathtt{x} = 1 \wedge \mathtt{emp}\}$

If we want the precondition $1 = 1$ (i.e. $\top$) then instantiate v to $\mathtt{x}$

$\{\mathtt{x} = \mathtt{x} \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ 1}\ \{\mathtt{x} = 1 \wedge \mathtt{emp}\}$

Heap reads axiom

Heap reads axiom:

$$\{\mathtt{x} = v_1 \wedge \mathbb{E} \mapsto v_2\}\ \mathtt{x\ :=\ [\mathbb{E}]}\ \{\mathtt{x} = v_2 \wedge \mathbb{E}[\mathtt{x}/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Heap reads axiom

Heap reads axiom:

$$\{x = v_1 \land \mathbb{E} \mapsto v_2\}\ \texttt{x := [}\mathbb{E}\texttt{]}\ \{x = v_2 \land \mathbb{E}[x/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Example (`x := [y]`)

σ

y	0xAB
x	2

h

0xAB	1

`x := [y]`

σ

y	0xAB
x	1

h

0xAB	1

Heap reads axiom

Heap reads axiom:

$$\{\mathtt{x} = v_1 \land \mathbb{E} \mapsto v_2\}\ \mathtt{x\ :=\ [}\mathbb{E}\mathtt{]}\ \{\mathtt{x} = v_2 \land \mathbb{E}[\mathtt{x}/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Example (`x := [y]`)

σ

y	0xAB
x	2

h

0xAB	1

`x := [y]`

σ

y	0xAB
x	1

h

0xAB	1

Heap read axiom instance:

$$\{\mathtt{x} = 2 \land \mathtt{y} \mapsto 1\}\ \mathtt{x\ :=\ [y]}\ \{\mathtt{x} = 1 \land \mathtt{y} \mapsto 1\}$$

Heap writes axiom

Heap writes axiom:

$$\{\mathbb{E}_1 \mapsto -\}\ [\mathbb{E}_1] := \mathbb{E}_2\ \{\mathbb{E}_1 \mapsto \mathbb{E}_2\}$$

where $(\mathbb{E}_1 \mapsto -)$ abbreviates $(\exists z. \mathbb{E}_1 \mapsto z)$ and z does no occur in $\mathbb{E}_1$

Heap writes axiom

Heap writes axiom:

$$\{\mathbb{E}_1 \mapsto -\}\ [\mathbb{E}_1] := \mathbb{E}_2\ \{\mathbb{E}_1 \mapsto \mathbb{E}_2\}$$

where $(\mathbb{E}_1 \mapsto -)$ abbreviates $(\exists z. \mathbb{E}_1 \mapsto z)$ and z does no occur in $\mathbb{E}_1$

Heap assignment semantics:

- evaluate expression $\mathbb{E}_1$ to get location l
- fault if location l is not in the current heap
- otherwise make the contents of location l the value of expression $\mathbb{E}_2$

Heap writes axiom

Heap writes axiom:

$$\{\mathbb{E}_1 \mapsto -\}\ [\mathbb{E}_1] := \mathbb{E}_2\ \{\mathbb{E}_1 \mapsto \mathbb{E}_2\}$$

where $(\mathbb{E}_1 \mapsto -)$ abbreviates $(\exists z.\mathbb{E}_1 \mapsto z)$ and z does no occur in $\mathbb{E}_1$

Heap assignment semantics:

- evaluate expression $\mathbb{E}_1$ to get location l
- fault if location l is not in the current heap
- otherwise make the contents of location l the value of expression $\mathbb{E}_2$

Example

$\{\mathrm{y} \mapsto -\}$ `[y] := 5` $\{\mathrm{y} \mapsto 5\}$

σ

y	0xAB

h

0xAB	-

`[y] := 5`

σ

y	0xAB

h

0xAB	5

Heap allocation axiom

Heap allocation axiom:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)\ \{\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]\}$$

where v is a variable diff. from $\mathtt{x}$ and not appearing in $\mathbb{E}_1, \ldots, \mathbb{E}_n$

Heap allocation assignment axiom means: if $\sigma(\mathtt{x}) = v$ and the heap is empty then executing $\mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)$ gives a heap consisting of n new consecutive locations, where location $\sigma(\mathtt{x}) + i$ contains $\sigma(\mathbb{E}_{i+1}[\mathtt{x}/v])$

Heap allocation axiom

Heap allocation axiom:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)\ \{\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]\}$$

where v is a variable diff. from $\mathtt{x}$ and not appearing in $\mathbb{E}_1, \ldots, \mathbb{E}_n$

Heap allocation assignment axiom means: if $\sigma(\mathtt{x}) = v$ and the heap is empty then executing $\mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)$ gives a heap consisting of n new consecutive locations, where location $\sigma(\mathtt{x}) + i$ contains $\sigma(\mathbb{E}_{i+1}[\mathtt{x}/v])$

σ

x	–
y	7

h

$\mathtt{x} := \mathtt{cons}(5, \mathtt{y} + 1)$

σ

x	$0xAB$
y	7

h

0xAB	5
0xAC	8

Heap allocation axiom

Heap allocation axiom:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)\ \{\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]\}$$

where v is a variable diff. from $\mathtt{x}$ and not appearing in $\mathbb{E}_1, \ldots, \mathbb{E}_n$

Heap allocation assignment axiom means: if $\sigma(\mathtt{x}) = v$ and the heap is empty then executing $\mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)$ gives a heap consisting of n new consecutive locations, where location $\sigma(\mathtt{x}) + i$ contains $\sigma(\mathbb{E}_{i+1}[\mathtt{x}/v])$

σ

x	–
y	7

h $\quad \mathtt{x} := \mathtt{cons}(5, \mathtt{y} + 1)$

σ

x	0x*AB*
y	7

h

0xAB	5
0xAC	8

$\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]$ abbreviates

$\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v]\ *\ (\mathtt{x}+1) \mapsto \mathbb{E}_2[\mathtt{x}/v]\ * \ldots *\ (\mathtt{x}+n-1) \mapsto \mathbb{E}_n[\mathtt{x}/v]$

Heap deallocation axiom

Heap deallocation axiom: $\{\mathbb{E} \mapsto -\}$ `dispose` $\mathbb{E}$ $\{\texttt{emp}\}$

where $(\mathbb{E} \mapsto -)$ abbreviates $(\exists z.\mathbb{E} \mapsto z)$ and z does no occur in $\mathbb{E}$

Heap deallocation axiom

Heap deallocation axiom: $\{\mathbb{E} \mapsto -\}$ `dispose` $\mathbb{E}$ $\{\mathtt{emp}\}$

where $(\mathbb{E} \mapsto -)$ abbreviates $(\exists z. \mathbb{E} \mapsto z)$ and z does no occur in $\mathbb{E}$

Heap deallocation: `dispose` $\mathbb{E}$

- evaluate $\mathbb{E}$ to get location l
- fault if location l is not in the current heap
- otherwise remove location l from the heap

Heap deallocation axiom means: if the heap is a singleton with domain $\sigma(\mathbb{E})$ then executing `dispose` $\mathbb{E}$ results in the empty heap.

Separation logic axioms - recap

Store assignment axiom:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x\ :=\ } \mathbb{E}\ \{\mathtt{x} = \mathbb{E}[\mathtt{x}/v] \wedge \mathtt{emp}\}$$

where v is an auxiliary variable which does not occur in $\mathbb{E}$

Heap reads axiom:

$$\{\mathtt{x} = v_1 \wedge \mathbb{E} \mapsto v_2\}\ \mathtt{x\ :=\ [}\mathbb{E}\mathtt{]}\ \{\mathtt{x} = v_2 \wedge \mathbb{E}[\mathtt{x}/v_1] \mapsto v_2\}$$

where v_1 and v_2 are auxiliary variables which do not occur in $\mathbb{E}$

Heap writes axiom: $\{\mathbb{E}_1 \mapsto -\}[\mathbb{E}_1] := \mathbb{E}_2\{\mathbb{E}_1 \mapsto \mathbb{E}_2\}$

where $(\mathbb{E}_1 \mapsto -)$ abbreviates $(\exists z.\mathbb{E}_1 \mapsto z)$ and z does no occur in $\mathbb{E}_1$

Heap allocation axiom:

$$\{\mathtt{x} = v \wedge \mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(\mathbb{E}_1, \ldots \mathbb{E}_n)\ \{\mathtt{x} \mapsto \mathbb{E}_1[\mathtt{x}/v], \ldots, \mathbb{E}_n[\mathtt{x}/v]\}$$

where v is a variable diff. from $\mathtt{x}$ and not appearing in $\mathbb{E}_1, \ldots, \mathbb{E}_n$

Heap deallocation axiom: $\{\mathbb{E} \mapsto -\}\ \mathtt{dispose}\ \ \mathbb{E}\ \{\mathtt{emp}\}$

where $(\mathbb{E} \mapsto -)$ abbreviates $(\exists z.\mathbb{E} \mapsto z)$ and z does no occur in $\mathbb{E}$

The frame rule

Frame rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{P * R\}\ \mathbb{C}\ \{Q * R\}}$$

where no variables modified by $\mathbb{C}$ appears free in R.

The Frame rule means that $\{P\}\ \mathbb{C}\ \{Q\}$ is restricted to the variables and parts of the heap that are actually used by $\mathbb{C}$.

The frame rule

Frame rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{P * R\}\ \mathbb{C}\ \{Q * R\}}$$

where no variables modified by $\mathbb{C}$ appears free in R.

Example

Is the following instance a legal instance of the Frame rule?
If so, why and if not, why not?

$$\frac{\{\texttt{emp}\}\ \texttt{x} := \texttt{cons}(1)\ \{\texttt{x} \mapsto 1\}}{\{\texttt{emp} \ * \ \texttt{x} \mapsto 1\}\ \texttt{x} := \texttt{cons}(1)\ \{\texttt{x} \mapsto 1 \ * \ \texttt{x} \mapsto 1\}}$$

The frame rule

Frame rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{P * R\}\ \mathbb{C}\ \{Q * R\}}$$

where no variables modified by $\mathbb{C}$ appears free in R.

Example

Is the following instance a legal instance of the Frame rule?
If so, why and if not, why not?

$$\frac{\{\mathtt{emp}\}\ \mathtt{x} := \mathtt{cons}(1)\ \{\mathtt{x} \mapsto 1\}}{\{\mathtt{emp}\ *\ \mathtt{x} \mapsto 1\}\ \mathtt{x} := \mathtt{cons}(1)\ \{\mathtt{x} \mapsto 1\ *\ \mathtt{x} \mapsto 1\}}$$

No, the command modifies $\mathtt{x}$ and R contains a free occurrence of $\mathtt{x}$.

SAT solvers

Propositional Logic

Formulas are defined by

$$\varphi \ ::= \ p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi$$

starting from propositional variables (atoms).

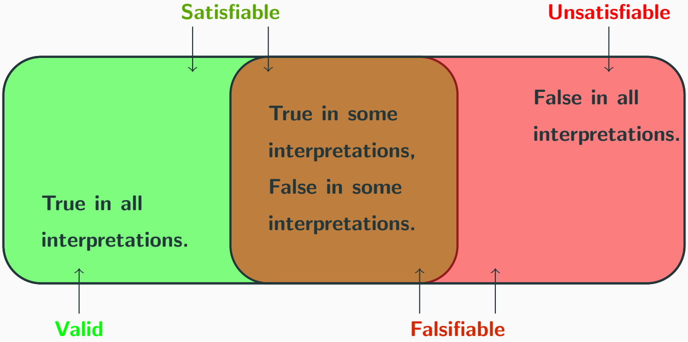
Interpretations assign truth values to propositional variables (true/false).

Further, we can compute the truth value of any formula (e.g., using truth tables).

A formula is satisfiable if there exists an interpretation which makes the formula true.

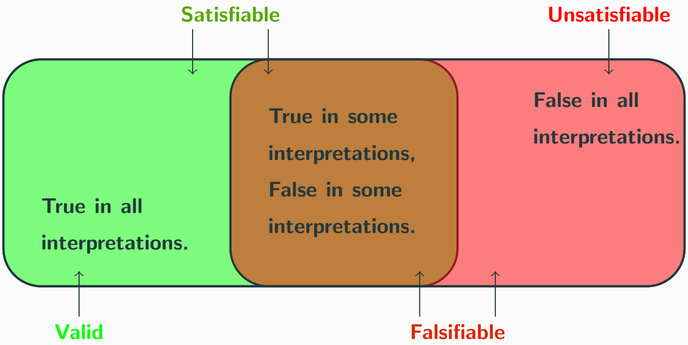
A formula is valid if is it true under all interpretations.

Satisfiability and Validity of formulas



$$
\begin{aligned}
\text{valid} &= \text{not falsifiable} \\
\text{satisfiable} &= \text{not unsatisfiable}
\end{aligned}
$$

Satisfiability and Validity of formulas



$$\begin{aligned} \text{valid} \quad &= \quad \text{not falsifiable} \\ \text{satisfiable} \quad &= \quad \text{not unsatisfiable} \end{aligned}$$

$$\boxed{\varphi \text{ satisfiable} \quad \text{iff} \quad \neg\varphi \text{ is falsifiable} \quad \text{iff} \quad \neg\varphi \text{ is not valid}}$$

$$\boxed{\varphi \text{ valid} \quad \text{iff} \quad \neg\varphi \text{ is unsatisfiable}}$$

The SAT problem

The SAT problem:

Given a propositional formula with n variables,
can we find an interpretation to make the formula true?

Is the formula satisfiable? If so, how?

The SAT problem

The SAT problem:

Given a propositional formula with n variables,
can we find an interpretation to make the formula true?

Is the formula satisfiable? If so, how?

The first known NP-complete problem, as proved by Stephen Cook in 1971.

The SAT problem

The SAT problem:

Given a propositional formula with n variables,
can we find an interpretation to make the formula true?

Is the formula satisfiable? If so, how?

The first known NP-complete problem, as proved by Stephen Cook in 1971.

Since SAT is NP-complete, is there hope?

SAT solvers

A SAT solver is a program that automatically decides whether a propositional formula is satisfiable (i.e, answers the SAT problem).

If it is satisfiable, a SAT solver will produce an example of an interpretation that satisfies the formula.

SAT solvers

A SAT solver is a program that automatically decides whether a propositional formula is satisfiable (i.e, answers the SAT problem).

If it is satisfiable, a SAT solver will produce an example of an interpretation that satisfies the formula.

Naive algorithm: enumerate all assignments to the n variables in the formula (2^n assignments!)

Worst case complexity is exponential (for all known algorithms).

SAT solvers

A SAT solver is a program that automatically decides whether a propositional formula is satisfiable (i.e, answers the SAT problem).

If it is satisfiable, a SAT solver will produce an example of an interpretation that satisfies the formula.

Naive algorithm: enumerate all assignments to the n variables in the formula (2^n assignments!)

Worst case complexity is exponential (for all known algorithms).

Perhaps surprisingly, many efficient SAT solvers exist!

- average cases encountered in practice can be handled (much) faster
- real problem instances will not be random: exploit implicit structure
- some variables will be tightly correlated with other variables
- some variables will be irrelevant for the difficult parts of the search

SAT solvers

- There are plenty of SAT solvers:
  - MiniSAT, PicoSAT
  - RelSAT
  - GRASP
  - ...
- There are also online SAT solvers:
  - Logictools
  - ...

SAT solvers

- There are plenty of SAT solvers:
  - MiniSAT, PicoSAT
  - RelSAT
  - GRASP
  - ...
- There are also online SAT solvers:
  - Logictools
  - ...
- SAT competition
  - The First International SAT Competition in 1992, followed by 1993, 1996, since 2002 every year, affiliated with the SAT conference

SAT solvers

- There are plenty of SAT solvers:
  - MiniSAT, PicoSAT
  - RelSAT
  - GRASP
  - ...
- There are also online SAT solvers:
  - Logictools
  - ...
- SAT competition
  - The First International SAT Competition in 1992, followed by 1993, 1996, since 2002 every year, affiliated with the SAT conference
- Early 90's: 100 variables, 200 clauses
- Today: 1,000,000 variables and 5,000,000 clauses.

Applications

Where can we find SAT technology today?

- Formal methods
  - Hardware model checking
  - Software model checking
  - Termination analysis of term-rewrite systems
  - Test pattern generation (testing of software & hardware)
  - ...
- Artificial intelligence
  - Planning
  - Knowledge representation
  - Games (n-queens, sudoku, ...)

Applications

Where can we find SAT technology today?

- Bioinformatics
  - Haplotype inference
  - Pedigree checking
  - ...
- Design automation
  - Equivalence checking
  - Fault diagnosis
  - Noise analysis
  - ...
- Security
  - Cryptanalysis
  - Inversion attacks on hash functions
  - ...

Applications

Where can we find SAT technology today?

- Computationally hard problems
  - Graph coloring
  - Traveling salesperson
  - ...
- Mathematical problems
  - van der Waerden numbers
  - Quasigroup open problems
  - ...
- Core engine for other solvers
- Integrated into theorem provers
  - HOL
  - Isabelle
  - ...

An example - Pythagorean Triples

Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then a, b, and c do not all have the same color?

An example - Pythagorean Triples

Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then a, b, and c do not all have the same color?

- Solution: nope
- for $n = 7825$ it is not possible
- the proof obtained by a SAT solver has 200 Terrabytes
- the largest Math proof ever (*see the article*)

An example - Pythagorean Triples

Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then a, b, and c do not all have the same color?

- Solution: nope
- for $n = 7825$ it is not possible
- the proof obtained by a SAT solver has 200 Terrabytes
- the largest Math proof ever *(see the article)*

How to encode this problem?

- for each integer i we have a Boolean variable x_i
- $x_i = 1$ if the color of i is 1 and $x_i = 0$ otherwise
- for each a, b, c such that $a^2 + b^2 = c^2$ we have two clauses:

$$\begin{aligned}(x_a \vee x_b \vee x_c) &\quad\sim\quad \neg(\neg x_a \wedge \neg x_b \wedge \neg x_c)\\ (\neg x_a \vee \neg x_b \vee \neg x_c) &\quad\sim\quad \neg(x_a \wedge x_b \wedge x_c)\end{aligned}$$

CNF - Conjunctive normal form

All current fast SAT solvers work on CNF (or slightly generalized CNF).

CNF - Conjunctive normal form

All current fast SAT solvers work on CNF (or slightly generalized CNF).

- A literal is a propositional variable or its negation
  - example: p, $\neg q$
  - For a literal l we write $\sim l$ for the negation of l cancelling double negations

CNF - Conjunctive normal form

All current fast SAT solvers work on CNF (or slightly generalized CNF).

- A literal is a propositional variable or its negation
  - example: p, $\neg q$
  - For a literal l we write $\sim l$ for the negation of l cancelling double negations
- A clause is a disjunction of literals
  - example: $p \lor \neg q \lor r$
  - Since $\lor$ is associative, we can represent clauses as lists of literals.
  - The empty clause (0 disjuncts) is defined to be $\perp$
  - A unit clause is a clause consisting of exactly one literal.

CNF - Conjunctive normal form

All current fast SAT solvers work on CNF (or slightly generalized CNF).

- A literal is a propositional variable or its negation
  - example: p, $\neg q$
  - For a literal l we write $\sim l$ for the negation of l cancelling double negations
- A clause is a disjunction of literals
  - example: $p \vee \neg q \vee r$
  - Since $\vee$ is associative, we can represent clauses as lists of literals.
  - The empty clause (0 disjuncts) is defined to be $\perp$
  - A unit clause is a clause consisting of exactly one literal.
- A formula is in CNF if it is a conjuction of clauses
  - example: $(p \vee \neg q \vee r) \wedge (\neg p \vee s \vee t \vee \neg u)$
  - Since $\wedge$ is associative, we can represent formulas in CNF as lists of clauses.
  - The empty conjunction is defined to be $\top$

CNF

We can represent CNF formulas as vectors of vectors of literals, which are often integers.

If "5" means x_5, then "-5" means $\neg x_5$.

Example

$[[2, -1], [3, -2, 1]]$

is the representation of the CNF formula

$(x_2 \vee \neg x_1) \wedge (x_3 \vee \neg x_2 \vee x_1)$

DIMACS Format

- the most common input format for SAT solvers
- a way to encode CNF formulas

Example
The input

```
c This is a comment
c This is another comment
p cnf 6 3
1 -2 3 0
2 4 5 0
4 6 0
```

represents the CNF formula $(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor x_4 \lor x_5) \land (x_4 \lor x_6)$

DIMACS Format

- At the beginning there can exist one or more comment line.
- Comment lines start with a `c`
- The following lines are information about the expression itself
- the Problem line starts with a `p`:

```
p FORMAT VARIABLES CLAUSES
```

- `FORMAT` should always be `cnf`
- `VARIABLES` is the number of variables in the expression
- `CLAUSES` is the number of clauses in the expression

Example
`p cnf 6 3` expresses that there are 6 variables and 3 clauses

DIMACS Format

- The next `CLAUSES` lines are for the clauses themselves
- Variables are enumerated from 1 to `VARIABLES`
- A negation is represented by $-$
- Each variable information is separated by a blank space
- A 0 is added at the end to mark the end of the clause

Example
`1 -2 3 0` expresses the clause $(x_1 \vee \neg x_2 \vee x_3)$

A planning problem

Example

Scheduling a meeting considering the following constraints:

- Adam can only meet on Monday and Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

A planning problem

Example

Scheduling a meeting considering the following constraints:

- Adam can only meet on Monday and Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

We represent week day *Monday*, *Tuesday*, ... as variables $x_1, x_2, \ldots$.

A planning problem

Example

Scheduling a meeting considering the following constraints:

- Adam can only meet on Monday and Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

We represent week day *Monday*, *Tuesday*, ... as variables $x_1, x_2, \ldots$.

We obtain the following formula in CNF:

$\varphi = (x_1 \vee x_3) \wedge (\neg x_3) \wedge (\neg x_5) \wedge (x_4 \vee x_5) \wedge \mathit{AtMostOne}$

A planning problem

Example (cont.)

$AtMostOne =$

$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5) \wedge$

$(\neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_5) \wedge$

$(\neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee \neg x_5) \wedge$

$(\neg x_4 \vee \neg x_5)$

A planning problem

Example (cont.)

$\varphi = (x_1 \vee x_3) \wedge (\neg x_3) \wedge (\neg x_5) \wedge (x_4 \vee x_5) \wedge AtMostOne$

$$AtMostOne = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_5) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee \neg x_5) \wedge (\neg x_4 \vee \neg x_5)$$

DIMACS Format c Planning a meeting
p cnf 5 14
1 3 0
-3 0
-5 0
4 5 0
-1 -2 0
-1 -3 0
-1 -4 0
-1 -5 0
-2 -3 0
-2 -4 0
-2 -5 0
-3 -4 0
-3 -5 0

-4 -5 0

Quiz time!

`https://www.questionpro.com/t/AT4NiZrmaJ`

References – Separation logic

- Lecture Notes on "Formal Methods for Software Engineering", Australian National University, Rajeev Goré.
- John Reynolds, "Introduction to Separation Logic (Preliminary Draft)", parts 1-4.
- Peter W. O'Hearn, "Resources, Concurrency and Local Reasoning", Theoretical Computer Science, Volume 375, Issue 1-3, Pages 271-307, 2007.