1. The Problem: The Design of Safe and Secure Computer-Based Systems

Software is Everywhere

- Exponential growth of hardware since 1975
  ⇒ exponential growth of software (favored by software engineering methods)
- Mainly manual activity ⇒ bugs are everywhere
Guaranteeing the Reliability and Security of Software-Intensive Systems

- A permanent objective since the origin of computer science
- An industrial requirement, in particular for safety and security critical software (validation can account for up to 60% of software development costs)

Validation/Formal Methods

- Bug-finding methods: unit, integration, and system testing, dynamic verification, bounded model-checking, error pattern mining, ...
- Absence of bug proving methods: formally prove that the semantics of a program satisfies a specification
  - theorem-proving & proof checking
  - model-checking
  - abstract interpretation
- In practice: complementary methods are used, very difficult to scale up

2. Abstract Interpretation

The Theory of Abstract Interpretation

- A theory of sound approximation of mathematical structures, in particular those involved in the behavior of computer systems
- Systematic derivation of sound methods and algorithms for approximating undecidable or highly complex problems in various areas of computer science
- Main practical application is on the safety and security of complex hardware and software computer systems
- Abstraction: extracting information from a system description that is relevant to proving a property
Applications of Abstract Interpretation

- **Static Program Analysis** [54], [59], [55] including Dataflow Analysis; [55], [58], Set-based Analysis [57], Predicate Abstraction [3], ... 
- **Grammar Analysis and Parsing** [6];
- **Hierarchies of Semantics and Proof Methods** [56], [5];
- **Typing & Type Inference** [53];
- **(Abstract) Model Checking** [58];
- **Program Transformation** (including program optimization, partial evaluation, etc) [12];

3. An Example of Theoretical Application: Semantics of the Eager λ-calculus


Applications of Abstract Interpretation (Cont’d)

- **Software Watermarking** [14];
- **Bisimulations** [71];
- **Language-based security** [63];
- **Semantics-based obfuscated malware detection** [70].
- **Databases** [50, 51, 52]
- **Computational biology** [60]
- **Quantum computing** [64, 68]

All these techniques involve sound approximations that can be formalized by **abstract interpretation**

Syntax
Syntax of the Eager $\lambda$-calculus

$$x, y, z, \ldots \in X$$ variables
$$c \in C$$ constants ($$X \cap C = \emptyset$$)
$$c ::= 0 | 1 | \ldots$$
$$v \in V$$ values
$$v ::= c | \lambda x.a$$
$$e \in E$$ errors
$$e ::= c | a | e a$$
$$a, a', a_1, \ldots, b, \ldots \in T$$ terms
$$a ::= x | v | a a'$$

Example I: Finite Computation

function argument

$$\left( (\lambda x\cdot x) \right) \left( (\lambda y\cdot y) \right) \left( (\lambda z\cdot z) \right) 0$$

$$\rightarrow$$ evaluate function

$$\left( (\lambda y\cdot y) \right) \left( (\lambda y\cdot y) \right) \left( (\lambda z\cdot z) \right) 0$$

$$\rightarrow$$ evaluate function, cont’d

$$\left( \lambda y\cdot y \right) \left( (\lambda z\cdot z) \right) 0$$

$$\rightarrow$$ evaluate argument

$$\left( \lambda y\cdot y \right) 0$$

$$\rightarrow$$ apply function to argument

$$0$$ a value!

Example II: Infinite Computation

function argument

$$\left( \lambda x\cdot x \right) \left( \lambda x\cdot x \right)$$

$$\rightarrow$$ apply function to argument

$$\left( \lambda x\cdot x \right) \left( \lambda x\cdot x \right)$$

$$\rightarrow$$ apply function to argument

$$\left( \lambda x\cdot x \right) \left( \lambda x\cdot x \right)$$

$$\rightarrow$$ apply function to argument

$$\ldots$$ non termination!
Example III: Erroneous Computation

\[
\begin{align*}
\text{function argument} & \quad \text{evaluate argument} \\
((\lambda x \cdot x x) ((\lambda z \cdot z) 0)) ((\lambda y \cdot y) 0) & \quad ((\lambda x \cdot x x) ((\lambda z \cdot z) 0)) 0 \\
& \quad ((\lambda x \cdot x x) 0) 0 \\
& \quad (0 0) 0 \\
\end{align*}
\]

a runtime error!

Traces
- \( T^* \) (resp. \( T^+ \), \( T^\omega \), \( T^\infty \) and \( T^\infty \)) be the set of finite (resp. nonempty finite, infinite, finite or infinite, and nonempty finite or infinite) sequences of terms
- \( \epsilon \) is the empty sequence \( \epsilon \cdot \sigma = \sigma \cdot \epsilon = \sigma \).
- \(|\sigma| \in \mathbb{N} \cup \{\omega\}\) is the length of \( \sigma \in T^\infty \). \(|\epsilon| = 0\).
- If \( \sigma \in T^+ \) then \(|\sigma| > 0 \) and \( \sigma = \sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma_{|\sigma|-1} \).
- If \( \sigma \in T^\omega \) then \(|\sigma| = \omega \) and \( \sigma = \sigma_0 \cdot \ldots \cdot \sigma_n \cdot \ldots \).

Operations on Traces
- For \( a \in T \) and \( \sigma \in T^\infty \), we define \( a \circ \sigma \) to be \( \sigma' \in T^\infty \) such that \( \forall i < |\sigma| : \sigma'_i = a \cdot \sigma_i \)

\[
\sigma = \sigma_0 \quad \sigma_1 \quad \sigma_2 \quad \sigma_3 \quad \ldots \quad \sigma_i \quad \ldots \\
\]

\[
a \circ \sigma = a \quad \sigma_0 \quad a \quad \sigma_1 \quad a \quad \sigma_2 \quad a \quad \sigma_3 \quad \ldots \quad a \quad \sigma_i \quad \ldots 
\]

- similarly \( \sigma \circ a \) is \( \sigma' \) where \( \forall i < |\sigma| : \sigma'_i = \sigma_i \cdot a \)

\[
\sigma = \sigma_0 \quad \sigma_1 \quad \sigma_2 \quad \sigma_3 \quad \ldots \quad \sigma_i \quad \ldots \\
\]

\[
\sigma \circ a = \sigma_0 \quad a \quad \sigma_1 \quad a \quad \sigma_2 \quad a \quad \sigma_3 \quad a \quad \ldots \quad \sigma_i \quad a \quad \ldots 
\]
Finite and Infinite Trace Semantics

Non-Standard Meaning of the Rules

The rules
\[ \mathcal{R} = \left\{ \frac{P_i}{C_i} \mid i \in \Delta \right\} \]
define
\[ \text{ifp} \subseteq F[\mathcal{R}] \]
where the consequence operator is
\[ F[\mathcal{R}](T) = \bigcup \left\{ C \mid P \subseteq T \land \frac{P}{C} \in \mathcal{R} \right\} \]
and ...

Bifinitary Trace Semantics $\mathcal{S}$ of the Eager $\lambda$-calculus\(^1\) [56]

Given $S, T \in \wp(\mathbb{T}^{\infty})$, we define
- $S^+ \triangleq S \cap \mathbb{T}^+$ \hspace{2cm} finite traces
- $S^\omega \triangleq S \cap \mathbb{T}^\omega$ \hspace{2cm} infinite traces
- $S \subseteq T \triangleq S^+ \subseteq T^+ \land S^\omega \supseteq T^\omega$ \hspace{0.5cm} computational order
- $\langle \wp(\mathbb{T}^{\infty}), \subseteq, T^\omega, T^+, \sqcup, \sqcap \rangle$ is a complete lattice

\(^1\) Note: $a[x \leftarrow b]$ is the capture-avoiding substitution of $b$ for all free occurrences of $x$ within $a$. We let $\text{PV}(a)$ be the free variables of $a$. We define the call-by-value semantics of closed terms (without free variables) $T^S (a \in T \setminus \text{PV}(a) = \emptyset)$.
Relational Semantics = α(Trace Semantics)
Abstraction to the Bifinitary Relational Semantics of the Eager $\lambda$-calculus

remember the input/output behaviors, forget about the intermediate computation steps

$$\alpha(T) \overset{\text{def}}{=} \{ \alpha(\sigma) \mid \sigma \in T \}$$

$$\alpha(\sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma_n) \overset{\text{def}}{=} \langle \sigma_0, \sigma_n \rangle$$

$$\alpha(\sigma_0 \cdot \ldots \cdot \sigma_n \cdot \ldots) \overset{\text{def}}{=} \langle \sigma_0, \bot \rangle$$

Natural Semantics

Bifinitary Relational Semantics of the Eager $\lambda$-calculus

$$v \Rightarrow v, \quad v \in V$$

$$a \Rightarrow \bot \quad \subseteq$$

$$a b \Rightarrow \bot \quad \subseteq, \quad a \in V$$

$$a | x \leftarrow v \Rightarrow r \quad \subseteq, \quad v \in V, \quad r \in V \cup \{\bot\}$$

$$(\lambda x \cdot a) \Rightarrow r$$

$$a \Rightarrow v, \quad v b \Rightarrow r \quad \subseteq, \quad v \in V, \quad r \in V \cup \{\bot\}$$

$$a b \Rightarrow r$$

$$b \Rightarrow v, \quad a v \Rightarrow r \quad \subseteq, \quad a \in V, \quad v \in V, \quad r \in V \cup \{\bot\}$$

$$a b \Rightarrow r$$

Natural Semantics = $\alpha$(Relational Semantics)
Abstraction to the Natural Big-Step Semantics of the Eager $\lambda$-calculus

remember the finite input/output behaviors, forget about non-termination

$$\alpha(T) \overset{\text{def}}{=} \bigcup \{ \alpha(\sigma) \mid \sigma \in T \}$$

$$\alpha(\langle \sigma_0, \sigma_n \rangle) \overset{\text{def}}{=} \{ \langle \sigma_0, \sigma_n \rangle \}$$

$$\alpha(\langle \sigma_0, \bot \rangle) \overset{\text{def}}{=} \emptyset$$

Natural Big-Step Semantics of the Eager $\lambda$-calculus [65]

$$v \Rightarrow v, \quad v \in V$$

$$a[x \leftarrow v] \Rightarrow r, \quad v \in V, \quad r \in V$$

$$\frac{(\lambda x \cdot a) \vDash v}{a \Rightarrow v}, \quad v \in V, \quad r \in V$$

$$\frac{a \Rightarrow v, \quad v \Rightarrow r}{a \cdot b \Rightarrow r}, \quad v \in V, \quad r \in V$$

$$\frac{b \Rightarrow v, \quad a \Rightarrow r}{a \cdot b \Rightarrow r}, \quad a \in V, \quad v \in V, \quad r \in V.$$
Abstraction to the Transition Semantics of the Eager \(\lambda\)-calculus

remember execution steps, for get about their sequencing

\[
\alpha(T) \overset{\text{def}}{=} \bigcup \{\alpha(\sigma) \mid \sigma \in T\}
\]

\[
\alpha(\sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma_n) \overset{\text{def}}{=} \{\langle \sigma_i, \sigma_{i+1} \rangle \mid 0 \leq i \wedge i < n\}
\]

\[
\alpha(\sigma_0 \cdot \ldots \cdot \sigma_n \cdot \ldots) \overset{\text{def}}{=} \{\langle \sigma_i, \sigma_{i+1} \rangle \mid i \geq 0\}
\]

Approximation

\[((\lambda x \cdot x) \cdot ((\lambda z \cdot z) 0)) (\lambda y \cdot y) \rightarrow ((\lambda x \cdot x) 0) (\lambda y \cdot y) \rightarrow (0 0) (\lambda y \cdot y)\]

an error!

Transition Semantics of the Eager \(\lambda\)-calculus [69]

\[((\lambda x \cdot a) v) \rightarrow a[x \leftarrow v]\]

\[
\begin{array}{c}
\text{a}_0 \rightarrow \text{a}_1 \\
\text{a}_0 \ \text{b} \rightarrow \text{a}_1 \ \text{b}
\end{array}
\]

\[
\begin{array}{c}
\text{b}_0 \rightarrow \text{b}_1 \\
\text{v} \ \text{b}_0 \rightarrow \text{v} \ \text{b}_1
\end{array}
\]

The Abstract Semantics are Correct by Calculational Design
4. Principle of Static Analysis

- Concrete Semantics

- Specification

(3.1) Abstract Semantics
5. An Example of Practical Application: The ASTRÉE Static Analyzer
Programs Analysed by ASTRÉE

- **Application Domain**: large safety critical embedded synchronous software (for real-time non-linear control of very complex control/command systems).

- **C programs**:
  - with
    - basic numeric datatypes, structures and arrays
    - pointers (including on functions),
    - floating point computations
    - tests, loops and function calls
    - limited branching (forward goto, break, continue)
  - with (cont’d)
    - union
    - pointer arithmetics & casts
  - **without**
    - dynamic memory allocation
    - recursive function calls
    - unstructured/backward branching
    - conflicting side effects
    - C libraries, system calls (parallelism)

*Such limitations are quite common for embedded safety-critical software.*
Concrete Semantics

Concrete Trace Semantics
- International norm of C (ISO/IEC 9899:1999)
- restricted by implementation-specific behaviors depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
- restricted by user-defined programming guidelines (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
- restricted by program specific user requirements (e.g. assert)

The Semantics of C is Hard (Ex. 1: Floats)

"Put $x$ in $[m, M] \mod (M - m)"$

$$x' = x - (\text{int})\left(\frac{(x-m)}{(M-m)}\right)(M-m);$$

- The programmer thinks $x' \in [m, M]$
- But with $m = 4095$, $M = -M$, IEEE double precision, and $x$ is the greatest float strictly less than $M$, then $x' = m - \epsilon$ ($\epsilon$ very small).

Floats are not real.

The Semantics of C is Hard (Ex. 2: Runtime Errors)
What is the effect of out-of-bounds array indexing?

% cat unpredictable.c
#include <stdio.h>
int main () { int n, T[1];
n = 2147483647;
printf("n = %i, T[n] = %i\n", n, T[n]);}

Yields different results on different machines:

<table>
<thead>
<tr>
<th>Machine</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macintosh PPC</td>
<td>2147483647</td>
</tr>
<tr>
<td>Macintosh Intel</td>
<td>-1208492044</td>
</tr>
<tr>
<td>PC Intel 32 bits</td>
<td>-135294988</td>
</tr>
<tr>
<td>PC Intel 64 bits</td>
<td>Bus error</td>
</tr>
</tbody>
</table>

Execution stops after a runtime error with unpredictable results².

² Equivalent semantics if no alarm.
Implicit Specification: Absence of Runtime Errors

- No violation of the norm of C (e.g. array index out of bounds, division by zero)
- No implementation-specific undefined behaviors (e.g. maximum short integer is 32767, NaN)
- No violation of the programming guidelines (e.g. static variables cannot be assumed to be initialized to 0)
- No violation of the programmer assertions (must all be statically verified).

Example: Dichotomy Search I

```c
% cat dichotomy.c
int main () {
  int R[100], X; short lwb, upb, m;
  lwb = 0; upb = 99;
  while (lwb <= upb) {
    m = upb + lwb;
    m = m » 1;
    if (X == R[m]) { upb = m; lwb = m+1; }
    else if (X < R[m]) { upb = m - 1; }
    else { lwb = m + 1; }
  }
  __ASTREE_log_vars((m));
}
```

Example: Dichotomy Search II

```c
% diff dichotomy.c dichotomy-bug.c
2,3c2,3
< int R[100], X; short lwb, upb, m;
< lwb = 0; upb = 99;
--
> int R[30000], X; short lwb, upb, m;
> lwb = 0; upb = 29999;
```

Astrée finds bugs in programs based on algorithms which have been formally proved correct.
Specification Can Be Tricky

- What is known about the execution environment?
- Warn on integer arithmetic overflows? Including left shifts (to extract bit fields)? Including in initializers?
- Warn on implicit cast/conversion? When they overflow?
- What is an incorrect access to a union field?
- ...
A “reasonable default choice” with analysis parameters for variants

[Superscript note: undefined except for unsigned to unsigned.]

Abstraction is Extremely Hard

- The analysis must be automatic (no user interaction)
- The abstraction must
  - ensure termination (and efficiency) of the analysis
  - be sound (ASTRÉE is a verifier, not a bug-finder)
  - scale up (100,000 to 1,000,000 LOCs)
  - be precise (no false alarm)

A grand challenge

General-Purpose Abstract Domains: Intervals and Octagons

Intervals:

\[
\begin{align*}
1 &\leq x \leq 9 \\
1 &\leq y \leq 20
\end{align*}
\]

Octagons [66]:

\[
\begin{align*}
1 &\leq x \leq 9 \\
x + y &\leq 77 \\
1 &\leq y \leq 20 \\
x - y &\leq 04
\end{align*}
\]

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [54, 66, 67]
Termination

SLAM uses CEGAR and does not terminate on

```c
% cat slam.c
int main() { int x, y;
    x = 0; y = 0;
    while (x < 2147483647)
        { x = x + 1; y = y + 1; }
    __ASTREE_assert((x == y));
}
```

whereas ASTÈRE uses widening/narrowing-based extrapolation techniques to prove the assertion

```c
% astree -exec-fn main slam.c |& egrep "WARN"
```

The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leaves

```c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;

typedef enum {F=0,T=1} BOOL;

BOOLEAN INIT; float P, X;

void filter () {
    static float E[2], S[2];
    if (INIT) {
        S[0] = X; P = X; E[0] = X;
    } else {
        P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
            + (S[0] * 1.5)) - (S[1] * 0.7));
    }
    E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
    /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}

void main () { X = 0.2 * X + 5; INIT = TRUE;
    while (1) {
        X = 0.9 * X + 35; /* simulated filter input */
        filter (); INIT = FALSE;
    }
}
```

2nd Order Digital Filter:

- Computes $X_n = \{ \alpha X_{n-1} + \beta X_{n-2} + Y_n \}
  \frac{I_n}{I_n}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.
Time Dependent Deviations [62]

% cat retro.c
typedef enum (FALSE=0, TRUE=1) BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev()
{ X=E;
  if (FIRST) { P = X; }
  else
  { P = (P - (((2.0 * P) - A) - B)
            * 4.491048e-03));
   B = A;
   if (SWITCH) {A = P;}
  else {A = X;}
  }
}

void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev();
    FIRST = FALSE;
    __ASTREE_wait_for_clock();
  }
%
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));

|P| <= (15. + 5.87747175411e-39
/ 1.19209290217e-07) * (1
+ 1.19209290217e-07) * clock
- 5.87747175411e-39
/ 1.19209290217e-07 <=
23.0393526881

Zero False Alarm Objective

Industrial constraints require ASTRÉE to be extremely precise:

- ASTRÉE is designed for a well-identified family of programs
- The analysis can be tuned using
  - parameters
  - analysis directives (which insertion can be automated)
  - extensions of the analyzer (by the tool designers)

Incompleteness

ASTRÉE does not know that

\[ \forall x, y \in \mathbb{Z} : 7y^2 - 1 \neq x^2 \]

so on the following program

void main() { int x, y;
  if ((-4681 <= y) && (y < 4681) && (x < 32767) && (-32767 < x) && ((7*y*y - 1) == x*x))
  { y = 1 / x; }
}

it produces a false alarm

% astree -exec-fn main repeat1.c | & egrep "WARN"
repeat1.c:5.8-13::[call#main@01::loop@4>=4.]: WARN: integer division by zero ([-32766, 32766]
and (1) / Z)

Example of directive

% cat repeat1.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
  int x = 100; BOOL b = TRUE;

  while (b) {
    x = x - 1;
    b = (x > 0);
  }
%
% astree -exec-fn main repeat1.c | & egrep "WARN"
repeat1.c:5.8-13::[call#main@02::loop@4>=4.]: WARN: signed int arithmetic range [-2147483649, 2147483646] not included in [-2147483648, 2147483647]
Example of directive (Cont’d)

```c
% cat repeat2.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
    int x = 100; BOOL b = TRUE;
    __ASTREE_boolean_pack((b,x));
    while (b) {
        x = x - 1;
        b = (x > 0);
    }
}
% astree –exec-fn main repeat2.c |& egrep "WARN"
%
The insertion of this directive could have been automated.
```

Application to Avionics Software

- **Primary flight control software**

- C program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)

- A340 family: 200,000 lines, A380: × 5

No false alarm, a world première!

---

Industrial Application

6. A Few Research Directions
Abstraction of Computations

– Semantics of concurrency (anticipated evolution of hardware)
– Abstract properties and specifications: safety, liveness, security, probabilistic behaviors, ...
– Time abstraction: continuous to discrete, scheduling, performance properties

Abstraction of Computational Paradigms

– Abstraction of data structures
– Abstraction of control structures: imperative, functional, procedural, logical, synchronous, parallel, distributed, and mobile control paradigms
– Abstraction of program structures: procedures, modules, objects, classes, ...
– Abstraction of communication and cooperation structures: synchronous/asynchronous lossy/lossless channels, events, semaphores, mobile communications, exogenous systems, ...

Abstraction Validation

– Abstraction translation: translation of abstractions while translating models (from mathematical models to programs)
– Verified abstractions: beyond toy examples

– Abstraction of hardware structures: memory caches, pipelines, branch prediction ... at the assembler level, hardware description languages
– Abstraction of biological systems: abstraction of agent-based descriptions of biological systems
Abstraction Automatization

- **Imprecision localization**: origin of false alarms
- **Automatic refinement**: automatic design of abstract domains to eliminate false alarms
- **Automatic abstraction**: too precise abstractions are costly

---

Abstract Interpretation

- Abstract interpretation is
  - a theory
  - with effective applications
  - and unprecedented industrial accomplishments.
- Further investigations of the theory are needed (while its scope of application broaden)
- The demand for applications is quasi-illimited

---

7. Conclusion

---

THE END, THANK YOU
8. Recent Publications


Invited Book Chapters


Refereed Journal Publications


Invited Conference or Workshop Proceedings Publications


\(^7\) The titles of the publications are clickable references to their web location, whenever available.


CS Colloquium, NYU, 9/4/2007 — 89 — P. Cousot


Recent Software

B. Blanchet, P. Cousot, R. Cousot, J. Peret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – The \textsc{astrée} analyser. – http://www.asterre.ens.fr/.

P. Cousot. – \textsc{anaa}: The abstract interpretation-based software watermarker, June 2003.


P. Cousot. – Abstract Interpretation: Theory and Practice, invited speaker. In: European Joint Conferences on Theory and Practice of Software (ETAPS’02), Grenoble, France, 8–12 April 2002.


Patents


Invited Conference Lectures and Tutorials


9. Other References


\[-a = (\lambda y \cdot y)\]
\[-\sigma = ((\lambda z \cdot z) 0) \cdot 0\]
\[-a \oplus \sigma =
(\lambda y \cdot y)((\lambda z \cdot z) 0) \cdot 0 =
((\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y) 0)\]

\[-\sigma = ((\lambda x \cdot x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot (\lambda y \cdot y)\]
\[-b = ((\lambda z \cdot z) 0)\]
\[-(\sigma \oplus b) =
((\lambda x \cdot x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot (\lambda y \cdot y)((\lambda z \cdot z) 0)) =
((\lambda x \cdot x) (\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y)((\lambda z \cdot z) 0))\]

\[-\sigma \cdot v \in \tilde{S}^+, (a \cdot v') \in \tilde{S}, (a \oplus \sigma) \cdot (a \cdot v') \in \tilde{S}, v, a \in V\]

\[-\sigma \cdot v = ((\lambda z \cdot z) 0) \cdot 0 \in \tilde{S}^+\]
\[-(a \cdot v') \cdot \sigma' = (\lambda y \cdot y) 0 \cdot 0 \in \tilde{S}\]
\[-(a \oplus \sigma) \cdot (a \cdot v') \cdot \sigma' =
((\lambda y \cdot y)((\lambda z \cdot z) 0) \cdot 0) \cdot 0 =
(\lambda y \cdot y)((\lambda z \cdot z) 0) \cdot (\lambda y \cdot y) 0 \cdot 0 \in \tilde{S}\]

\[-\sigma \cdot v = ((\lambda x \cdot x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot (\lambda y \cdot y) \in \tilde{S}^+\]
\[-(v \cdot b') \cdot \sigma' = (\lambda y \cdot y)((\lambda z \cdot z) 0) \cdot (\lambda y \cdot y) 0 \cdot 0 \in \tilde{S}\]
\[-(\sigma \oplus b) \cdot (v \cdot b') \cdot \sigma' =
(((\lambda x \cdot x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot ((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot (\lambda y \cdot y) 0 \cdot 0 =
((\lambda x \cdot x) (\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y)((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y) 0 \cdot 0 \in \tilde{S}\]