# Abstract Interpretation, Temporal Logic and Data Flow Analysis

**Patrick COUSOT**
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05
France
mailto:Patrick.Cousot@ens.fr
http://www.dmi.ens.fr/~cousot

**Radhia COUSOT**
CNRS & École Polytechnique
LIX
91440 Palaiseau cedex
France
mailto:rcousot@lix.polytechnique.fr
http://lix.polytechnique.fr/~radhia

Dagstuhl Seminar 99151 on Program Analysis, April 11–16, 1999



## 1. Introduction



## Motivation

- Claims:
  - In model-checking the properties to be checked are a user-defined parameter of the model checker;
  - In abstract interpretation, the properties to be discovered are wired in the (generic) static program analyzer;
- Not completely true (see the invariant and intermittent assertions for abstract testing in [1, 2]);
- Can we do better? temporal logic specifications.

**References**

[1] F. Bourdoncle. *Abstract Debugging of Higher-Order Imperative Languages.* ACM PLDI'93, 46–55, 1993.

[2] P. Cousot. *Semantic foundations of program analysis.* In S.S. Muchnick and N.D. Jones, eds, *Program Flow Analysis: Theory and Applications*, ch. 10, 303–342. Prentice-Hall, 1981.



## Objective of the talk

- Not a general presentation;
- Just consider a very simple example:

  Derive a dataflow analysis by abstract interpretation of its temporal logic specification

# 2. Background



## Traditional Dataflow Analysis… [3] [1, 2]

- In traditional boolean dataflow analysis, the problem specification for the flowchart program is left informal;
- Or, it is expressed informally along a program path and then there is a merge over all paths [3];
- Correctness by intuition (informal arguments).

**Reference**

[3] T.J. Marlowe and B.G. Ryder. *Properties of data flow frameworks: A unified model.* Acta Informatica 28, 121–163, 1990.

[1] small part of it only!
[2] for short, oversimplified!
[3] which coincide with fixpoint solutions for distributive frameworks.



## Abstract Interpretation coming in… [4]

- A prefix-closed path semantics of the transition system (program) is expressed in fixpoint form;
- The dataflow problem specification is by an abstraction function describing:
  - The property along one path;
  - How path properties are merged;
- Using abstract interpretation techniques, the boolean dataflow fixpoint equations were formally derived by calculational design from the trace-based semantics. Correctness by construction.
- Only one example (available expressions).

**Reference**

[4] P. Cousot and R. Cousot. *Systematic design of program analysis frameworks.* $6^{\text{th}}$ ACM POPL, 269–282, 1979.



## Model-checking coming in… [5]

- The program is a flowchart (with obvious semantics);
- Abstract interpretation is used to derive the transfer functions at the node level;
- The dataflow problem specification is by a branching time temporal logic formula;
- Classical model-checking algorithms are used to check the program model for the temporal formula;
- Correctness by specification.

**Reference**

[5] B. Steffen. *Data flow analysis as model-checking.* TACS'91, 346–364, Springer, 1991.

## Model-checking with more Abstract Interpretation... [6]

- The abstract flowchart is proved to be an abstract interpretation of a trace-based semantics;
- The dataflow problem specification is by a branching time temporal logic formula;
- It is shown that the algorithms checking the program model for the temporal formula *yield the same result as* the dataflow equations;
- Correctness by abstract interpretation (flowchart) and by specification (dataflow problem).

**Reference**

[6] D.A. Schmidt. *Data-flow analysis is model checking of abstract interpretations.* 25th ACM POPL, 1998.



## Live Variables Analysis is Unsound

**Reference**

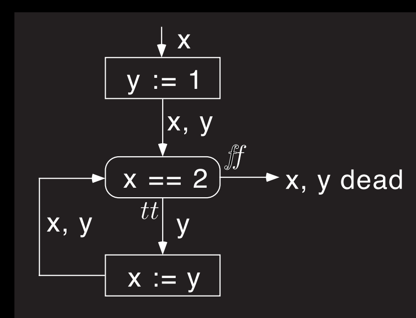
[7] D.A. Schmidt. *Data-flow analysis is model checking of abstract interpretations.* 25th ACM POPL, 1998 (see Figure 5 and paragraph 7 "Why Some Analyses are Unsound").



## Bad news...

**A Bug!**






Liveness analysis claims y to be live before test which is wrong when initially not x == 2! [8]

**Reference**

[8] D.A. Schmidt. *Data-flow analysis is model checking of abstract interpretations.* 25th ACM POPL, 1998.

## Questions...

- What should we think of a model-checking based design methodology which let you design unsound analyzes[7]?

- **Who is guilty?**
  - Abstract interpretation[8]?
  - Data flow analysis[9]?
  - Model-checking?

[7] is everybody as wise as D. Schmidt to find a 25 years old bug? Does B. Steffen tool in Passau effectively signals that bug to the user?
[8] I can't believe it!
[9] D. Schmidt forgives them by claiming "Of course, data-flow practitioners are well aware of the above problem, and disaster does not arise in practice ... But we might not be so fortunate in general."

## Which (general) fix cures the problem in the context of program analysis?

## My Diagnosis...

- Model-checking is **guilty!**
- The lacuna is that the **model-checking** specification by a temporal logic formula **does not take the abstraction process into account**;
- This is common in the model-checking community:
  - who really cares about how the finite model is obtained?
  - the model is the truth, the specification is the truth, model-checking is only about their concordance!
- In the program analysis community we (should) care: the programming language semantics is the referential (or should be).

## Good news...[10]

**Abstract Interpretation!**

[10] I am sure you got it right!

## In the rest of the talk, I will explain...

- How to design a dataflow analysis specified by a temporal formula and an abstraction so that it is correct by construction;
- To do so I just have to show that:

**Model-checking is an abstract interpretation!**

and then instanciate for the dataflow analysis problem specified by a temporal formula;



## 3. RTL: Reversible Temporal Logic



## In the rest of the talk, I will explain...

- How to design a dataflow analysis specified by a temporal formula and an abstraction so that it is correct by construction;
- To do so I just have to show that:

**Model-checking is an abstract interpretation!**

and then instanciate for the dataflow analysis problem specified by a temporal formula;
- Only a very small part is shown (indeed only what is necessary for live variables analysis);



## Which Temporal Logic?

- Dataflow analysis people are used to reason on (merge over all) paths so we prefer a linear time logic (one path at the time) to the branching time logics considered by Steffen and Schmidt;
- Dataflow analysis people make no essential distinction beetween forward and backward analyses so that one should directly derive one from the other [15]; We introduce a new temporal reversal operator to make past and future completely symmetric;
- Dataflow analysis people make no essential distinction beetween minimal and maximal flow problem so that one should directly derive one from the other (using duality [16]).

[15] as in P. Cousot & R. Cousot *Systematic design of program analysis frameworks.* 6th ACM POPL, 269–282, 1979 where backward is just forward for the inverse transition system.

[16] as in P. Cousot. Semantic foundations of program analysis. S.S. Muchnick & N.D. Jones, eds, *Program Flow Analysis: Theory and Applications*, ch 10, 303–342. Prentice-Hall, 1981.

## RTL: Reversible Temporal Logic — Syntax

| | | | |
|---|---|---|---|
| $l \in \mathcal{L}$ | | | *locations* |
| $\pi \in \mathcal{S}$ | | | *state predicates* |
| $\pi$ | $::=$ | $\mathrm{tt}$ | true |
| | $\mid$ | $\mathrm{ff}$ | false |
| | $\mid$ | $\mathsf{at}(l)$ | at control predicate |
| | $\mid$ | $\ldots$ | |
| $\tau \in \mathcal{T}$ | | | *transition predicates* |
| $\tau$ | $::=$ | $\mathbf{1}$ | identity |
| | $\mid$ | ${\tau_1}^{-1}$ | inverse |
| | $\mid$ | $\ldots$ | |



## RTL: Reversible Temporal Logic — Syntax (continued)

| | | | |
|---|---|---|---|
| $\ldots$ | | | |
| $\varphi, \psi \in \mathcal{F}$ | | | *temporal formulae* |
| $\varphi$ | $::=$ | $\pi$ | state predicate |
| | $\mid$ | $\tau$ | transition predicate |
| | $\mid$ | $\bigcirc \varphi_1$ | next |
| | $\mid$ | $\varphi_1 \mathbf{U} \varphi_2$ | until |
| | $\mid$ | ${\varphi_1}^{\curvearrowleft}$ | reversal |
| | $\mid$ | $\varphi_1 \vee \varphi_2$ | disjunction |
| | $\mid$ | $\neg \varphi_1$ | negation |



## RTL: Reversible Temporal Logic — Semantic Domains

| | |
|---|---|
| $\Sigma$ | set of states |
| $\mathbb{S} \triangleq \wp(\Sigma)$ | semantic domain of state predicates |
| $\mathbb{T} \triangleq \wp(\Sigma \times \Sigma)$ | semantic domain of transition predicates |
| $\mathbb{P} \triangleq \mathbb{Z} \mapsto \Sigma$ | paths |
| $\mathbb{C} \triangleq \mathbb{Z} \times \mathbb{P}$ | computations |
| $\mathbb{F} \triangleq \wp(\mathbb{C})$ | semantic domain of temporal formulae |



## RTL: Reversible Temporal Logic — Semantic functions

| | |
|---|---|
| $\mathfrak{S} \in \mathcal{S} \mapsto \mathbb{S}$ | semantics of state predicates |
| $\in \mathcal{S} \mapsto \Sigma \mapsto \mathbb{B}$ | (isomorphic alternative) |
| $\mathfrak{T} \in \mathcal{T} \mapsto \mathbb{T}$ | semantics of transition predicates |
| $\in \mathcal{T} \mapsto (\Sigma \times \Sigma) \mapsto \mathbb{B}$ | (isomorphic alternative) |
| $\mathfrak{F} \in \mathcal{F} \mapsto \mathbb{F}$ | semantics of temporal formulae |

## RTL: Reversible Temporal Logic — Semantics

$$
\begin{aligned}
\mathfrak{S}[\![\mathrm{tt}]\!] &\triangleq \Sigma \\
\mathfrak{S}[\![\mathrm{ff}]\!] &\triangleq \emptyset \\
&\ldots \\
\mathfrak{T}[\![\mathbf{1}]\!] &\triangleq \{\langle s,\, s\rangle \mid s \in \Sigma\} \\
\mathfrak{T}[\![\tau^{-1}]\!] &\triangleq (\mathfrak{T}[\![\tau]\!])^{-1} \\
&\ldots \\
\mathfrak{F}[\![\pi]\!] &\triangleq \{\langle i,\, \sigma\rangle \mid \sigma_i \in \mathfrak{S}[\![\pi]\!]\} \\
\mathfrak{F}[\![\tau]\!] &\triangleq \{\langle i,\, \sigma\rangle \mid \langle \sigma_i,\, \sigma_{i+1}\rangle \in \mathfrak{T}[\![\tau]\!]\}
\end{aligned}
$$



## RTL: Reversible Temporal Logic — Abbreviations

| | | | |
|---|---|---|---|
| $\varphi_1 \wedge \varphi_2$ | $\triangleq$ | $\neg(\neg\varphi_1 \vee \neg\varphi_2)$ | conjunction |
| $\varphi_1 \rightarrow \varphi_2$ | $\triangleq$ | $\neg(\varphi_1) \vee \varphi_2$ | implication |
| $\Diamond\varphi$ | $\triangleq$ | $\mathrm{tt}\ \mathbf{U}\ \varphi$ | sometime or eventually |
| $\Box\varphi$ | $\triangleq$ | $\neg\Diamond\neg\varphi$ | always or henceforth |
| $\varphi_1 \Rightarrow \varphi_2$ | $\triangleq$ | $\Box(\varphi_1 \rightarrow \varphi_2)$ | entailment |
| $\varphi_1\ \mathbf{W}\ \varphi_2$ | $\triangleq$ | $(\varphi_1\ \mathbf{U}\ \varphi_2) \vee \Box\varphi_1$ | unless or waiting-for |
| $\ominus\varphi$ | $\triangleq$ | $(\bigcirc\varphi\ )$ | previous |
| $\varphi_1\ \mathbf{S}\ \varphi_2$ | $\triangleq$ | $(\varphi_1\ \ \mathbf{U}\ \varphi_2\ )$ | since |
| $\boxminus\varphi$ | $\triangleq$ | $(\Box\varphi\ )$ | has always been |
| $\diamondminus\varphi$ | $\triangleq$ | $(\Diamond\varphi\ )$ | once |
| $\varphi_1\ \mathbf{B}\ \varphi_2$ | $\triangleq$ | $(\varphi_1\ \ \mathbf{W}\ \varphi_2\ )$ | back to |



## RTL: Reversible Temporal Logic — Semantics (Continued)

$$
\begin{aligned}
\mathfrak{F}[\![\bigcirc\varphi_1]\!] &\triangleq \{\langle i,\, \sigma\rangle \mid \langle i+1,\, \sigma\rangle \in \mathfrak{F}[\![\varphi_1]\!]\} \\
\mathfrak{F}[\![\varphi_1\ \mathbf{U}\ \varphi_2]\!] &\triangleq \{\langle i,\, \sigma\rangle \mid \exists k \geq i : \langle k,\, \sigma\rangle \in \mathfrak{F}[\![\varphi_2]\!] \wedge \\
&\qquad\qquad \forall j : i \leq j < k : \langle j,\, \sigma\rangle \in \mathfrak{F}[\![\varphi_1]\!]\} \\
\mathfrak{F}[\![\varphi_1 \vee \varphi_2]\!] &\triangleq \mathfrak{F}[\![\varphi_1]\!] \cup \mathfrak{F}[\![\varphi_2]\!] \\
\mathfrak{F}[\![\neg\varphi]\!] &\triangleq \mathfrak{C} \setminus \mathfrak{F}[\![\varphi]\!] \qquad \text{(also written } \neg\mathfrak{F}[\![\varphi]\!]) \\
\mathfrak{F}[\![\varphi^{\cdot}]\!] &\triangleq \{\langle -i,\, \lambda j \cdot \sigma_{-j}\rangle \mid \langle i,\, \sigma\rangle \in \mathfrak{F}[\![\varphi]\!]\} \\
&\quad \text{(also written } (\mathfrak{F}[\![\varphi]\!])^{\cdot})
\end{aligned}
$$



## Implication and Equivalence of Temporal Formulae

| | | | |
|---|---|---|---|
| $\varphi_1 \Rrightarrow \varphi_2$ | $\triangleq$ | $\mathfrak{F}[\![\varphi_1]\!] \subseteq \mathfrak{F}[\![\varphi_2]\!]$ | Implication |
| $\varphi_1 \equiv \varphi_2$ | $\triangleq$ | $\mathfrak{F}[\![\varphi_1]\!] = \mathfrak{F}[\![\varphi_2]\!]$ | Equivalence |

($\equiv$ is a congruence on $\mathcal{F}$).

## RTL: Reversible Temporal Logic — Fixpoint Semantics

$$\mathfrak{F}[\varphi_1 \mathbf{U} \varphi_2] = \mathrm{lfp}^{\subseteq} \lambda X \cdot \mathfrak{F}[\varphi_2] \cup (\mathfrak{F}[\varphi_1] \cap \mathrm{pre}[X]) \qquad (1)$$

where

$$\mathrm{pre}[X] \triangleq \{\langle i-1,\, \sigma\rangle \mid \langle i,\, \sigma\rangle \in X\}$$

so that $\mathfrak{F}[\bigcirc \varphi] = \mathrm{pre}[\mathfrak{F}[\varphi]]$ whence:

$$\varphi_1 \mathbf{U} \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \mathbf{U} \varphi_2))$$



## RTL: Reversible Temporal Logic — Fixpoint Semantics (Continued)

All other cases directly follows by:

- Lattice duality:

$$\mathrm{gfp}^{\sqsubseteq} F = \mathrm{lfp}^{\sqsupseteq} F$$

- Park negation duality[17]:

$$\neg(\mathrm{gfp}\, F) = \mathrm{lfp}\, \lambda X \cdot \neg F(\neg X)$$

- Reversal duality:

$$(\mathrm{lfp}\, F)^{\curvearrowleft} = \mathrm{lfp}\, \lambda X \cdot (F(X^{\curvearrowleft}))^{\curvearrowleft}$$

[17] in the complete boolean lattice



## 4. The Semantics of Programs



## Small Step Operational Semantics

Transition predicate:

$$\mathit{trans} \in \mathcal{T}$$

Final states are the only possible blocking states and they are repeated forever:

$$\mathit{final} \triangleq \neg\, \mathit{trans} \wedge \sqsubset \mathbf{1}$$

and symmetrically for the initial states:

$$\mathit{init} \triangleq \ominus(\neg\, \mathit{trans}) \wedge \sqsubseteq \mathbf{1}$$

## Trace-Based Operational Semantics

- We define the forward and backward transition predicates as:

$$\begin{aligned} \mathit{ftrans}, \mathit{btrans} &\in \mathcal{T} \\ \mathit{ftrans} &\stackrel{\triangle}{=} \mathit{trans} \vee \mathit{final} \\ \mathit{btrans} &\stackrel{\triangle}{=} \ominus\, \mathit{trans} \vee \mathit{init} \end{aligned}$$

- The trace-based operational semantics of a program is (including non-termination and symmetrically non intialization):

| | |
|---|---|
| $\mathit{fsem} \stackrel{\triangle}{=} \sqsubset \mathit{ftrans}$ | forward operational semantics |
| $\mathit{bsem} \stackrel{\triangle}{=} \sqsubseteq \mathit{btrans}$ | backward operational semantics |
| $\mathit{sem} \stackrel{\triangle}{=} \mathit{fsem} \wedge \mathit{bsem}$ | operational semantics |



## In one definition...

We use the following notation:

$$\langle M,\ \preceq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L,\ \sqsubseteq\rangle$$

for Galois connections:

$$\forall x \in M, y \in L : \alpha(x) \sqsubseteq y \iff x \preceq \gamma(y)$$

We write $\langle M,\ \preceq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\twoheadrightarrow}} \langle L,\ \sqsubseteq\rangle$ when $\alpha$ is surjective, $\langle M,\ \preceq\rangle \underset{\alpha}{\overset{\gamma}{\twoheadleftarrow\!\!\!\!\rightarrow}} \langle L,\ \sqsubseteq\rangle$ when $\alpha$ is injective and $\langle M,\ \preceq\rangle \underset{\alpha}{\overset{\gamma}{\twoheadleftarrow\!\!\!\!\twoheadrightarrow}} \langle L,\ \sqsubseteq\rangle$ when $\alpha$ is bijective.



## 5. Abstract Interpretation



## ... and one theorem

**Theorem 1** *If $\langle M,\ \preceq,\ 0,\ \vee\rangle$ is a cpo, the pair $\langle \alpha,\ \gamma\rangle$ is a Galois connection $\langle M,\ \preceq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L,\ \sqsubseteq\rangle$, $\mathcal{F} \in M \overset{\text{mon}}{\longmapsto} M$ and $\mathcal{G} \in L \overset{\text{mon}}{\longmapsto} L$ are monotonic and satisfy the* semi-commutation condition

$$\forall y \in L : \gamma(y) \preceq \operatorname{lfp}^{\preceq} \mathcal{F} \Longrightarrow \alpha \circ \mathcal{F} \circ \gamma(y) \sqsubseteq \mathcal{G}(y)$$

*or equivalently* $\forall x \in M : \gamma \circ \alpha(x) \preceq \operatorname{lfp}^{\preceq} \mathcal{F} \Rightarrow \alpha \circ \mathcal{F}(x) \sqsubseteq \mathcal{G} \circ \alpha(x)$

*or equivalently* $\forall y \in L : \gamma(y) \preceq \operatorname{lfp}^{\preceq} \mathcal{F} \Longrightarrow \mathcal{F} \circ \gamma(y) \preceq \gamma \circ \mathcal{G}(y)$

*then*

$$\operatorname{lfp}^{\preceq} \mathcal{F} \preceq \gamma(\operatorname{lfp}^{\sqsubseteq} \mathcal{G})$$

*and equivalently* $\alpha(\operatorname{lfp}^{\preceq} \mathcal{F}) \sqsubseteq \operatorname{lfp}^{\sqsubseteq} \mathcal{G}$.

## 6. The Model-Checking Abstractions



## Boolean Universal Abstraction

The *boolean universal satisfaction abstraction* $\boldsymbol{\alpha}^{\forall}{}_{[\varphi]}(M)$ checks all computations of the model $M$ for the temporal formula $\varphi$:

$$\begin{array}{rcl|rcl} & \boldsymbol{\alpha}^{\forall} \in \mathcal{F} \mapsto \wp(\Sigma) \mapsto \mathbb{B} & & & \boldsymbol{\gamma}^{\forall} \in \mathcal{F} \mapsto \mathbb{B} \mapsto \wp(\Sigma) & \\ \boldsymbol{\alpha}^{\forall}{}_{[\varphi]}(M) & \triangleq & M \models \varphi & \boldsymbol{\gamma}^{\forall}{}_{[\varphi]}(b) & \triangleq & (b \mathrel{?} [\![\varphi]\!] \mathrel{¿} \Sigma) \\ & = & M \subseteq [\![\varphi]\!] & & & \end{array}$$

This is a generic Galois connection parameterized by the temporal formula $\varphi$:

$$\langle \wp(\Sigma), \subseteq\rangle \underset{\boldsymbol{\alpha}^{\forall}{}_{[\varphi]}}{\overset{\boldsymbol{\gamma}^{\forall}{}_{[\varphi]}}{\leftrightarrows}} \langle \mathbb{B}, \Longleftarrow\rangle$$



## By Dualization: Four Different Abstractions

| | | |
|---|---|---|
| $\boldsymbol{\alpha}^{\forall}{}_{[\varphi]}(M) \triangleq M \subseteq [\![\varphi]\!]$ | $\langle \wp(\Sigma), \subseteq\rangle \underset{\boldsymbol{\alpha}^{\forall}{}_{[\varphi]}}{\overset{\boldsymbol{\gamma}^{\forall}{}_{[\varphi]}}{\leftrightarrows}} \langle \mathbb{B}, \Longleftarrow\rangle$ | Universal |
| $\boldsymbol{\alpha}^{\exists}{}_{[\varphi]}(M) \triangleq \neg\,\boldsymbol{\alpha}^{\forall}{}_{[\neg\varphi]}(M)$ | $\langle \wp(\Sigma), \subseteq\rangle \underset{\boldsymbol{\alpha}^{\exists}{}_{[\varphi]}}{\overset{\boldsymbol{\gamma}^{\exists}{}_{[\varphi]}}{\leftrightarrows}} \langle \mathbb{B}, \Longrightarrow\rangle$ | Existential |
| $\boldsymbol{\alpha}^{\tilde\forall}{}_{[\varphi]}(M) \triangleq \neg\,\boldsymbol{\alpha}^{\forall}{}_{[\varphi]}(\neg M)$ | $\langle \wp(\Sigma), \supseteq\rangle \underset{\boldsymbol{\alpha}^{\tilde\forall}{}_{[\varphi]}}{\overset{\boldsymbol{\gamma}^{\tilde\forall}{}_{[\varphi]}}{\leftrightarrows}} \langle \mathbb{B}, \Longrightarrow\rangle$ | Dual Universal |
| $\boldsymbol{\alpha}^{\tilde\exists}{}_{[\varphi]}(M) \triangleq \neg\,\boldsymbol{\alpha}^{\exists}{}_{[\varphi]}(\neg M)$ | $\langle \wp(\Sigma), \supseteq\rangle \underset{\boldsymbol{\alpha}^{\tilde\exists}{}_{[\varphi]}}{\overset{\boldsymbol{\gamma}^{\tilde\exists}{}_{[\varphi]}}{\leftrightarrows}} \langle \mathbb{B}, \Longleftarrow\rangle$ | Dual Existential |



## More on the Existential Satisfaction Abstraction

$$\begin{aligned} \boldsymbol{\alpha}^{\exists}{}_{[\varphi]}(M) &\triangleq \neg\,\boldsymbol{\alpha}^{\forall}{}_{[\neg\varphi]}(M) \\ &= \ldots \qquad\qquad \text{easing calculation} \\ &= (M \cap [\![\varphi]\!]) \neq \emptyset \end{aligned} \tag{2}$$

The *boolean existential satisfaction abstraction* $\boldsymbol{\alpha}^{\exists}{}_{[\varphi]}(M)$ checks that some computations of the model $M$ do satisfy the temporal formula $\varphi$.

## State Static Partitionning

- Let's check the model for each state;
- State projection:

$$\bullet_{\downarrow\bullet} \in \mathcal{M} \mapsto \Sigma \mapsto \mathcal{M}$$

$$M_{\downarrow s} \triangleq \{\langle i,\, \sigma\rangle \in M \mid \sigma_i = s\}$$

- Static state partitionning abstraction:

$$\boldsymbol{\alpha}_\Sigma(M) \triangleq \lambda s \cdot M_{\downarrow s} \qquad \boldsymbol{\gamma}_\Sigma(S) \triangleq \bigcup_{s\in\Sigma} S(s)_{\downarrow s}$$

- Galois isomorphism [18]:

$$\langle \mathcal{M},\, \subseteq\rangle \underset{\boldsymbol{\alpha}_\Sigma}{\overset{\boldsymbol{\gamma}_\Sigma}{\leftrightarrows}} \langle \prod_{s\in\Sigma} \mathcal{M}_{\downarrow s},\, \dot{\subseteq}\rangle$$

[18] so no information is lost.



## State Partionned Satisfaction Abstractions

$$\dot{\boldsymbol{\alpha}}^{\forall}_{\Sigma\varphi} \triangleq \boldsymbol{\alpha}^{\dot\forall}_{\varphi} \circ \boldsymbol{\alpha}_\Sigma \qquad \langle \mathcal{M},\, \subseteq\rangle \underset{\dot{\boldsymbol{\alpha}}^{\forall}_{\Sigma\varphi}}{\overset{\dot{\boldsymbol{\gamma}}^{\forall}_{\Sigma\varphi}}{\leftrightarrows}} \langle \Sigma \mapsto \mathcal{B},\, \dot{\Leftarrow}\rangle$$
$$\dot{\boldsymbol{\gamma}}^{\forall}_{\Sigma\varphi} \triangleq \boldsymbol{\gamma}_\Sigma \circ \boldsymbol{\gamma}^{\dot\forall}_{\varphi}$$

$$\dot{\boldsymbol{\alpha}}^{\tilde\forall}_{\Sigma\varphi} \triangleq \boldsymbol{\alpha}^{\dot{\tilde\forall}}_{\varphi} \circ \widetilde{\boldsymbol{\alpha}}_\Sigma \text{ [19]} \qquad \langle \mathcal{M},\, \supseteq\rangle \underset{\dot{\boldsymbol{\alpha}}^{\tilde\forall}_{\Sigma\varphi}}{\overset{\dot{\boldsymbol{\gamma}}^{\tilde\forall}_{\Sigma\varphi}}{\leftrightarrows}} \langle \Sigma \mapsto \mathcal{B},\, \dot{\Rightarrow}\rangle$$
$$\dot{\boldsymbol{\gamma}}^{\tilde\forall}_{\Sigma\varphi} \triangleq \widetilde{\boldsymbol{\gamma}}_\Sigma \circ \boldsymbol{\gamma}^{\dot{\tilde\forall}}_{\varphi}$$

$$\dot{\boldsymbol{\alpha}}^{\exists}_{\Sigma\varphi} \triangleq \boldsymbol{\alpha}^{\dot\exists}_{\varphi} \circ \boldsymbol{\alpha}_\Sigma \qquad \langle \mathcal{M},\, \subseteq\rangle \underset{\dot{\boldsymbol{\alpha}}^{\exists}_{\Sigma\varphi}}{\overset{\dot{\boldsymbol{\gamma}}^{\exists}_{\Sigma\varphi}}{\leftrightarrows}} \langle \Sigma \mapsto \mathcal{B},\, \dot{\Rightarrow}\rangle$$
$$\dot{\boldsymbol{\gamma}}^{\exists}_{\Sigma\varphi} \triangleq \boldsymbol{\gamma}_\Sigma \circ \boldsymbol{\gamma}^{\dot\exists}_{\varphi}$$

$$\dot{\boldsymbol{\alpha}}^{\tilde\exists}_{\Sigma\varphi} \triangleq \boldsymbol{\alpha}^{\dot{\tilde\exists}}_{\varphi} \circ \widetilde{\boldsymbol{\alpha}}_\Sigma \qquad \langle \mathcal{M},\, \supseteq\rangle \underset{\dot{\boldsymbol{\alpha}}^{\tilde\exists}_{\Sigma\varphi}}{\overset{\dot{\boldsymbol{\gamma}}^{\tilde\exists}_{\Sigma\varphi}}{\leftrightarrows}} \langle \Sigma \mapsto \mathcal{B},\, \dot{\Leftarrow}\rangle$$
$$\dot{\boldsymbol{\gamma}}^{\tilde\exists}_{\Sigma\varphi} \triangleq \widetilde{\boldsymbol{\gamma}}_\Sigma \circ \boldsymbol{\gamma}^{\dot{\tilde\exists}}_{\varphi}$$

[19] $\widetilde{\alpha_\Sigma}$ is the dual of $\alpha_\Sigma$.



## Location Static Partitionning

- Let's now have a *coarser partition* according to *locations* (program points, call strings, contours, ... [20]);
- The locations are assumed to cover all states:

$$\forall s \in \Sigma : \exists l \in \mathcal{L} : s \in \mathcal{S}[\mathsf{at}(l)] \tag{3}$$

[20] need not be finite!



## Location Static Partitionning Abstraction

- Location partitionning:

$$\boldsymbol{\alpha}^{\forall}_{\mathcal{L}} \triangleq \lambda f \cdot \lambda l \cdot \bigwedge_{s\in\mathcal{S}[\mathsf{at}(l)]} f(s) \qquad \langle \Sigma \mapsto \mathcal{B},\, \dot{\Leftarrow}\rangle \underset{\boldsymbol{\alpha}^{\forall}_{\mathcal{L}}}{\overset{\boldsymbol{\gamma}^{\forall}_{\mathcal{L}}}{\leftrightarrows}} \langle \mathcal{L} \mapsto \mathcal{B},\, \dot{\Leftarrow}\rangle$$

$$\boldsymbol{\gamma}^{\forall}_{\mathcal{L}} \triangleq \lambda g \cdot \lambda s \cdot \bigvee_{s\in\mathcal{S}[\mathsf{at}(l)]} g(l)$$

- Dually $\boldsymbol{\alpha}^{\exists}_{\mathcal{L}}$ and $\boldsymbol{\gamma}^{\exists}_{\mathcal{L}}$ such that:

$$\langle \Sigma \mapsto \mathcal{B},\, \dot{\Rightarrow}\rangle \underset{\boldsymbol{\alpha}^{\exists}_{\mathcal{L}}}{\overset{\boldsymbol{\gamma}^{\exists}_{\mathcal{L}}}{\leftrightarrows}} \langle \mathcal{L} \mapsto \mathcal{B},\, \dot{\Rightarrow}\rangle$$

## Location Partionned Satisfaction Abstractions

$$
\begin{array}{ll}
\dot{\boldsymbol\alpha}^{\forall}_{\ddot{\mathcal{L}}\varphi} \triangleq \boldsymbol\alpha^{\forall}_{\mathcal{L}} \circ \dot{\boldsymbol\alpha}^{\forall}_{\ddot{\Sigma}\varphi} & \langle\ , \subseteq\rangle \xleftrightarrow[\dot{\alpha}^{\forall}_{\ddot{\mathcal{L}}\varphi}]{\dot{\gamma}^{\forall}_{\ddot{\mathcal{L}}\varphi}} \langle \mathcal{L} \mapsto\ , \stackrel{\cdot}{\Longleftarrow}\rangle \\
\dot{\boldsymbol\gamma}^{\forall}_{\ddot{\mathcal{L}}\varphi} \triangleq \dot{\boldsymbol\gamma}^{\forall}_{\ddot{\Sigma}\varphi} \circ \boldsymbol\gamma^{\forall}_{\mathcal{L}} & \\
\dot{\boldsymbol\alpha}^{\tilde\forall}_{\ddot{\mathcal{L}}\varphi} \triangleq \boldsymbol\alpha^{\exists}_{\mathcal{L}} \circ \dot{\boldsymbol\alpha}^{\tilde\forall}_{\ddot{\Sigma}\varphi} & \langle\ , \supseteq\rangle \xleftrightarrow[\dot{\alpha}^{\tilde\forall}_{\ddot{\mathcal{L}}\varphi}]{\dot{\gamma}^{\tilde\forall}_{\ddot{\mathcal{L}}\varphi}} \langle \mathcal{L} \mapsto\ , \stackrel{\cdot}{\Longrightarrow}\rangle \\
\dot{\boldsymbol\gamma}^{\tilde\forall}_{\ddot{\mathcal{L}}\varphi} \triangleq \dot{\boldsymbol\gamma}^{\tilde\forall}_{\ddot{\Sigma}\varphi} \circ \boldsymbol\gamma^{\exists}_{\mathcal{L}} & \\
\dot{\boldsymbol\alpha}^{\exists}_{\ddot{\mathcal{L}}\varphi} \triangleq \boldsymbol\alpha^{\exists}_{\mathcal{L}} \circ \dot{\boldsymbol\alpha}^{\exists}_{\ddot{\Sigma}\varphi} & \langle\ , \subseteq\rangle \xleftrightarrow[\dot{\alpha}^{\exists}_{\ddot{\mathcal{L}}\varphi}]{\dot{\gamma}^{\exists}_{\ddot{\mathcal{L}}\varphi}} \langle \mathcal{L} \mapsto\ , \stackrel{\cdot}{\Longrightarrow}\rangle \\
\dot{\boldsymbol\gamma}^{\exists}_{\ddot{\mathcal{L}}\varphi} \triangleq \dot{\boldsymbol\gamma}^{\exists}_{\ddot{\Sigma}\varphi} \circ \boldsymbol\gamma^{\exists}_{\mathcal{L}} & \\
\dot{\boldsymbol\alpha}^{\tilde\exists}_{\ddot{\mathcal{L}}\varphi} \triangleq \boldsymbol\alpha^{\forall}_{\mathcal{L}} \circ \dot{\boldsymbol\alpha}^{\tilde\exists}_{\ddot{\Sigma}\varphi} & \langle\ , \supseteq\rangle \xleftrightarrow[\dot{\alpha}^{\tilde\exists}_{\ddot{\mathcal{L}}\varphi}]{\dot{\gamma}^{\tilde\exists}_{\ddot{\mathcal{L}}\varphi}} \langle \mathcal{L} \mapsto\ , \stackrel{\cdot}{\Longleftarrow}\rangle \\
\dot{\boldsymbol\gamma}^{\tilde\exists}_{\ddot{\mathcal{L}}\varphi} \triangleq \dot{\boldsymbol\gamma}^{\tilde\exists}_{\ddot{\Sigma}\varphi} \circ \boldsymbol\gamma^{\forall}_{\mathcal{L}} &
\end{array}
\tag{4}
$$



## 7. The Calculational Design of the Abstract Model-Checking Algorithms by Abstract Interpretation



## Location Partionned Existential Satisfaction Abstractions of the Forward Trace Semantics

- Assume we are interested in checking for any location $l \in \mathcal{L}$ whether there is a computation from $l$ such that $\tau_1$ will hold until eventually $\tau_2$ does hold;
- Formally we want to calculate/compute:

$$\dot{\boldsymbol\alpha}^{\exists}_{\ddot{\mathcal{L}}[\tau_1 \mathbf{U} \tau_2]}(\mathfrak{F}[\![\mathit{fsem}]\!]) \tag{5}$$

- Design strategy:
  - Express this as the abstraction of a fixpoint (see (1))
  - Use the fixpoint approximation Theorem 1 with the Galois connection (4) (or dual forms).



## Fixpoint Abstraction

$$
\begin{aligned}
& \dot{\boldsymbol\alpha}^{\exists}_{\ddot{\mathcal{L}}[\tau_1 \mathbf{U} \tau_2]}(\mathfrak{F}[\![\mathit{fsem}]\!]) \\
= \ & \ldots \quad \text{skipping 12 lines of hand computation using (1)} \\
= \ & \dot{\boldsymbol\alpha}^{\exists}_{\ddot{\mathcal{L}}[\mathit{fsem}]}(\mathrm{lfp}^{\subseteq}\ \lambda X \cdot \mathfrak{F}[\![\tau_2]\!] \cup (\mathfrak{F}[\![\tau_1]\!] \cap \mathrm{pre}[X]))
\end{aligned}
$$

## Calculating the Semi-commuting Abstract Transformer

We assume:

$$\begin{aligned} X &= \mathfrak{F}[\tau_2] \cup (\mathfrak{F}[\tau_1] \cap \mathrm{pre}[X]) \\ \Psi &= \mathit{ftrans} \wedge \bigcirc \Psi \end{aligned}$$

and calculate:

$$\begin{aligned} & \dot{\boldsymbol{\alpha}}^{\exists}_{\ddot{\mathcal{L}}[\mathit{fsem}]}(\mathfrak{F}[\tau_2] \cup (\mathfrak{F}[\tau_1] \cap \mathrm{pre}[X])) \\ \dot{\Longrightarrow} \; & \ldots \quad \text{skipping 25 lines of hand computation} \\ = \; & F^{\sharp}(\dot{\boldsymbol{\alpha}}^{\exists}_{\ddot{\mathcal{L}}[\mathit{fsem}]}(X)) \end{aligned}$$

so that Theorem 1 with the Galois connection (4), we conclude .../...




.../...

$$\begin{aligned} \dot{\boldsymbol{\alpha}}^{\exists}_{\ddot{\mathcal{L}}[\tau_1 \mathbf{U} \tau_2]}(\mathfrak{F}[\mathit{fsem}]) &= \dot{\boldsymbol{\alpha}}^{\exists}_{\ddot{\mathcal{L}}[\mathit{fsem}]}(\mathrm{lfp}^{\subseteq}\, \lambda X \cdot \mathfrak{F}[\tau_2] \cup (\mathfrak{F}[\tau_1] \cap \mathrm{pre}[X])) \\ &\dot{\Longrightarrow} \mathrm{lfp}^{\dot{\Longrightarrow}}\, F^{\sharp} \end{aligned}$$

where

$$F^{\sharp}(X) \triangleq \lambda l \cdot (\exists l' : \mathfrak{T}^{\exists}_{\mathcal{L}}[\tau_2](l, l')) \vee \bigvee_{l' \in \mathrm{succ}(l)} \mathfrak{T}^{\exists}_{\mathcal{L}}[\tau_1](l, l') \wedge X(l')$$

and

$$\mathrm{succ}(l) \triangleq \{l' \mid \exists s \in \mathfrak{S}[\mathsf{at}(l)] : \exists s' \in \mathfrak{S}[\mathsf{at}(l')] : \mathfrak{T}[\mathit{ftrans}](s, s')\}$$

$$\mathfrak{T}^{\exists}_{\mathcal{L}}[\tau](l, l') \triangleq \bigvee_{s \in \mathfrak{S}[\mathsf{at}(l)] \wedge \mathfrak{T}[\mathit{ftrans}](s, s') \wedge s' \in \mathfrak{S}[\mathsf{at}(l')]} \mathfrak{T}[\tau](s, s')$$




## 8. Application to Live-Variables Analysis

"In *live-variable* analysis we wish to know for variable x and point $p$ whether the value of x at $p$ could be used along some path in the flow graph starting at $p$. If so, we say x is *live* at $p$; otherwise x is *dead* at $p$" [9, p. 631].

References

[9] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers. Principles, Technique and Tools.* Addison-Wesley, 1986.




## Live-Variables Analysis is a Sound Location Partionned Model-Checking Existential Abstraction of the Trace Semantics

- For a single flowchart node:

  mod(x) : transitions potentially modifying variable x
  used(x) : transitions definitively using the value of variable x

- Along one path: Variable x is live at the origin of a computation iff it will not be modified until it is used:

$$\mathsf{isLive}(\mathtt{x}) \triangleq (\neg\, \mathsf{mod}(\mathtt{x}))\ \mathbf{U}\ \mathsf{used}(\mathtt{x})$$

- Merge over some path: Variable x is live at location $l$ if and only if it is live on some computation path starting from that location:

$$\mathit{Live}(\mathtt{x}) \triangleq \dot{\boldsymbol{\alpha}}^{\exists}_{\ddot{\mathcal{L}}[\mathsf{isLive}(\mathtt{x})]}(\mathfrak{F}[\mathit{fsem}])$$

## The Classical Live-Variables Dataflow Equations are Definitely Sound

$$
\begin{aligned}
Live(\mathtt{x}) \; \dot{\Longrightarrow} \; & Live^\sharp(\mathtt{x}) \\
& \triangleq \; \mathrm{lfp}^{\dot{\Longrightarrow}} \; \lambda X \cdot \lambda l \cdot \; (\exists l' : \mathfrak{T}^{\exists}_{\mathcal{L}}[\![\mathsf{used}(\mathtt{x})]\!](l, l')) \; \vee \\
& \qquad\qquad \bigvee_{l' \in \mathrm{succ}(l)} \mathfrak{T}^{\exists}_{\mathcal{L}}[\![(\neg\, \mathsf{mod}(\mathtt{x}))]\!](l, l') \wedge X(l')
\end{aligned}
$$

- Note: $\dot{\Longrightarrow}$, not $\dot{\Longleftarrow}$!
- Hence $Dead^\sharp(\mathtt{x}) \triangleq \neg\, Live^\sharp(\mathtt{x}) \; \dot{\Longrightarrow} \; \neg\, Live(\mathtt{x}) \triangleq Dead(\mathtt{x})$;
- **So Dave forgives the practitionners!**
- **And now the practitionners forgive Dave!**



## 9. Conclusion



## So what is Unsound?

- Live-variables analysis is a location partionned model-checking existential/merge over some paths abstraction of the trace semantics;
- It is unsound to reason on live-variables analysis as if it were a location partionned model-checking universal/merge over all paths abstraction of the trace semantics;
- Model-checking is unsound in that it does not make explicit which of the abstractions is involved (and there are many such as $\dot{\alpha}^{\forall}_{\ddot{\mathcal{L}}}$, $\dot{\alpha}^{\tilde{\forall}}_{\ddot{\mathcal{L}}}$, $\dot{\alpha}^{\exists}_{\ddot{\mathcal{L}}}$ and $\dot{\alpha}^{\tilde{\exists}}_{\ddot{\mathcal{L}}}$).



## Conclusion (on Live-Variables Analysis Being Unsound)

- Data-flow analysis was not guilty;
- Model-Checking was guilty, by not taking the abstraction process into account;
- Abstract Interpretation was the rescuer:
  - Which abstractions are used is made explicit;
  - Abstraction is used for the formal calculational design of correct algorithms;
  - Abstract Interpretation provides an unambiguous understanding of what is going on.

## Conclusion (on Model-Checking Design by Abstract Interpretation)

- In this talk, we have chosen to handle a striking example rather than present formally the full theory;
- In full generality, we have to handle any $\boldsymbol{\alpha}_{\llbracket\varphi\rrbracket}(\mathfrak{F}\llbracket\psi\rrbracket)$ for $\boldsymbol{\alpha}$ in the cube of abstractions and all possible RTL formulae $\varphi, \psi \in \mathcal{F}$ (to get interleaved fixpoints);
- A more general and debatable question:

  **Is model-checking of any practical use in program static analysis?** [23]

[23] In G. Nelson talk the invariant were 10% of the program size. Who will write the invariants for his 30000 lines Java program in the form of a 3000 lines temporal formula? Isn't abstract debugging/testing with invariant/intermittent assertions (virtually) included in the program text much better?




## Abstract Interpretation...

In one single formal framework, abstract interpretation lets you meta-understand the foundational aspects of:

- Data flow analysis;
- Constraint based program analysis;
- Types and effect systems;
- ...
- Relationships between semantics;
- ...




## A final advertising...

**What can abstract interpretation do for you?**




## Abstract Interpretation...

In one single formal framework, abstract interpretation lets you meta-understand the foundational aspects of:

- Data flow analysis;
- Constraint based program analysis;
- Types and effect systems;
- ...
- Relationships between semantics;
- ...

and now:

- **Model-Checking;**
- **Including its use in the design of sound program analyzes!**

## Abstract Interpretation…

Abstract interpretation is a theory of discrete approximation of semantics, not only a peculiar static program analysis method.



## More on the Calculational Design of Abstract Interpretations

A complete calculational design of an abstract interpreter:

P. Cousot.
*Calculational System Design.*
chapter "The Calculational Design of a Generic Abstract Interpreter".
NATO ASI Series F. IOS Press, Amsterdam, 1999.

and its OCAML implementation:

P. Cousot.
The Marktoberdorf'98 generic abstract interpreter.
`http://www.dmi.ens.fr/~cousot/Marktoberdorf98.shtml`
November 1998.



# THE END