Bridging static and dynamic program analysis using fuzzy logic

Jacob Lidman & Josef Svenningsson
Chalmers University of Technology
{lidman, josefs}@chalmers.se

Static program analysis is used to summarize properties over all dynamic executions. In a unifying approach based on 3-valued logic properties are either assigned a definite value or unknown. But in summarizing a set of executions, a property is more accurately represented as being biased towards true, or towards false. Compilers use program analysis to determine benefit of an optimization. Since benefit (e.g., performance) is justified based on the common case understanding bias is essential in guiding the compiler. Furthermore, successful optimization also relies on understanding the quality of the information, i.e. the plausibility of the bias. If the quality of the static information is too low to form a decision we would like a mechanism that improves dynamically.

We consider the problem of building such a reasoning framework and present the fuzzy data-flow analysis. Our approach generalize previous work that use 3-valued logic. We derive fuzzy extensions of data-flow analyses used by the lazy code motion optimization and unveil opportunities previous work would not detect due to limited expressiveness. Furthermore we show how the results of our analysis can be used in an adaptive classifier that improve as the application executes.

1 Introduction

How can one reconcile static and dynamic program analysis? These two forms of analysis complement each other: static analysis summarizes all possible runs of a program and thus provide soundness guarantees, while dynamic analysis provides information about the particular runs of a program which actually happen in practice and can therefore provide more relevant information. Being able to combine these two paradigms has applications on many forms on analyses, such as alias analysis [16, 21] and dependence analysis [18].

Compilers use program analysis frameworks to prove *legality* as well as determining *benefit* of transformations. Specifications for legality are composed of *safety* and *liveness* assertions (i.e. universal and existentially quantified properties), while specifications for benefit use assertions that hold in the *common case*. This reason for adopting the common case is that few transformations improve performance in general (i.e., for every input, environment). Similarly most transformations could potentially improve performance in a least one case. As such, compiler optimizations are instead motivated based on (an approximation of) the majority case, i.e. the (weighted) mean. While determining legality has improved due to advances in the verification community the progress in establishing benefit has been slow.

In this paper we introduce fuzzy data-flow analysis, a framework for static program analysis based on fuzzy logic. The salient feature of our framework is that it can naturally incorporate dynamic information while still being a static analysis. This ability comes thanks to a shift from "crisp" sets where membership is binary, as employed in conventional static analysis, to fuzzy sets where membership is gradual.

Submitted to:
QAPL 2017

We make the following contributions:

- Section 3 introduces our main contribution, the fuzzy data-flow framework.
- Section 4 demonstrates the benefit of our framework by presenting a generalization of a well-known code motion algorithm and we show how this generalization provides new opportunities for optimizations previous approaches would not discover.
- Section 4 shows how fuzzy logic can benefit program analysis by (1) using second-order fuzzy sets to separate uncertainty in data-flow and control-flow and hence improve an inter-procedural analysis and (2) using fuzzy regulators to refine the results of our static analysis, hence improving the precision dynamically.

2 Preliminaries

We introduce and define fuzzy sets and the operators that form fuzzy logic. These concepts will be used in Section 3 to define the transfer functions of our data-flow analysis.

2.1 Fuzzy set

Elements of a crisp set[1] are either members or non-members w.r.t to a universe of discourse. A fuzzy set (FS) instead allow partial membership denoted by a number from the unit interval $[0,1]$. The membership degree typically denotes vagueness. The process to convert crisp membership to fuzzy grades is called *fuzzification* and the inverse is called *defuzzification*. Following Dubois et al. [9, 8] let S be a crisp set and $\mu : S \mapsto [0,1]$ a *membership function* (MF) then $\langle S,\mu\rangle$ is a fuzzy set. As a convention, if S is understood from context we sometimes refer to μ as a fuzzy set. The membership function formalizes the fuzzification. Fuzzy sets are ordered point-wise, i.e. $(S,\mu_A) \leq (S,\mu_B) \Leftrightarrow \forall s \in S\colon\ \mu_A(s) \leq \mu_B(s)$.

We can accommodate some notion about uncertainty of vagueness by considering a type-2 fuzzy set where the membership degree itself is a fuzzy set. Type-2 FS (T2FS) membership functions are composed of a primary (J_s) and secondary (μ) membership $\{\langle (s,u), \mu(s,u)\rangle \mid s \in S, u \in J_s \subseteq [0,1]\}$. Here uncertainty is represented by the secondary membership that define the possibility of the primary membership. When for each x and u, it holds $\mu(x,u) = 1$ the T2FS is called an *interval* T2FS. Gehrke et al. [10] showed that this can equivalently be described as an interval valued fuzzy sets (IVFS) where $\mu\colon S \to \{[l,u] \,|\, \bot \leq l \leq u \leq \top\}$. IVFS are a special case of lattice valued fuzzy sets (L-fuzzy sets) where the membership domain forms a lattice over $[0,1]$. Defuzzification of T2FS often proceeds in two phases. The first phase applies *type reduction* to transform the T2FS to a type-1 FS (T1FS). The second phase then applies a type-1 defuzzification.

2.2 Fuzzy logic

Fuzzy logic defines many-valued formal systems to reason about truth in the presence of vagueness. Contrary to classical logic the law of excluded middle ($p \vee \neg p = \top$) and the law of non-contradiction ($p \wedge \neg p = \bot$) does not, in general, hold for these systems. Fuzzy logic uses T-, S- and C- norms to generalize the logical operators $\wedge$, $\vee$ and $\neg$. We compactly represent a fuzzy logic by $\langle \tilde{\wedge}, \tilde{\vee}, \tilde{\neg}\rangle$[2] which is sometimes called a *De Morgan system* [9] because it satisfies a generalization of De Morgans laws: $\tilde{\neg}(P \,\tilde{\wedge}\, Q) \Leftrightarrow \tilde{\neg} P \,\tilde{\vee}\, \tilde{\neg} Q$ and $\tilde{\neg}(P \,\tilde{\vee}\, Q) \Leftrightarrow \tilde{\neg} P \,\tilde{\wedge}\, \tilde{\neg} Q$.

[1]In the context of fuzzy logic, crisp or Boolean set refer to a classical set to avoid confusion with fuzzy sets.

[2]Although one would expect the definition of a fuzzy logic to include a "fuzzy implication" operator in this work we do not consider it.

	Fuzzy logic	**T-norm**	**S-norm**	**C-norm**
1	*Min-Max*	$\min(x,y)$	$\max(x,y)$	$1-x$
2	*Algebraic Sum-product*	xy	$x+y-xy$	$1-x$
3	*Lukasiewicz*	$max(x+y-1,0)$	$min(x+y,1)$	$1-x$
4	*Nilpotent*	$\begin{cases} min(x,y) & x+y>1 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} max(x,y) & x+y<1 \\ 1 & \text{otherwise} \end{cases}$	$1-x$

Table 1: Common instantiations of fuzzy logics

Definition 1. *Let U be a binary function $[0,1]^2 \to [0,1]$ that is commutative, associative and increasing and has an identity element $e \in [0,1]$. If $e = 1$ then U is a **Triangular norm (T-norm)** and if $e = 0$ then U is a **Triangular conorm (S-norm)**[3].*

Definition 2. *A **C-norm** is a unary function $n\colon [0,1] \to [0,1]$ that is decreasing, involutory (i.e., $n(n(x)) = x$) with boundary conditions (i.e, $n(0) = 1, n(1) = 0$).*

Standard examples of fuzzy logics are shown in Table 1 [9, 8]. Examples 1-3 are special cases (and limits) of the Frank family of fuzzy logics that are central to our work and formally defined in Definition 3.

Definition 3. *Let $s \in [0,1] \cup \{\infty\}$ then the **Frank family** of T-norms is defined by:*

$$T^s(x,y) = \begin{cases} \min(x,y) & s = 0 \\ xy & s = 1 \\ \max(x+y-1,0) & s = \infty \\ log_s\left(1 + \frac{(s^x-1)(s^y-1)}{s-1}\right) & otherwise \end{cases}$$

The set of intervals in $[0,1]$ forms a bounded partial order $\langle \mathbb{I}, \sqsubseteq, \top, \bot \rangle$[4] where $[l_x, u_x] \leq [l_y, u_y] \Leftrightarrow (l_x \leq l_y) \wedge (u_x \leq u_y)$, $\top = [1,1]$ and $\bot = [0,0]$. As per Gehrke et al. [10] we can point-wise lift a T1FS fuzzy logic $\langle \tilde{\wedge}, \tilde{\vee}, \tilde{\neg} \rangle$ to a IVFS fuzzy logic, i.e., $[l_x, u_x] \odot [l_y, u_y] = [l_x \odot l_y, u_x \odot u_y], \hat{\odot} \in \{\tilde{\wedge}, \tilde{\vee}\}$ and $\tilde{\neg}[l,u] = [\tilde{\neg} u, \tilde{\neg} l]$.

3 Fuzzy data-flow analysis

Static data-flow analyses deduce values of semantic properties that are satisfied the dynamics of the application. The dynamics is formalized as a system of monotone transfer functions and collector functions. Transfer functions describe how blocks alter the semantic properties. Collectors functions merge results from different, possibly mutual exclusive, branches of the application. The solution of the analysis is obtained through Kleene iteration and is a unique fixed-point of the system of equations. In a classical framework the domain of the values is binary, i.e. either true (1) or false (0). The interpretation of these values depends on the type of analysis. The value true means that the property can possibly hold in a *may*-analysis (i.e., it is impossible that the value is always false) while it means that the property always holds in a *must*-analysis. The value false could mean either the opposite of true or that the result is inconclusive.

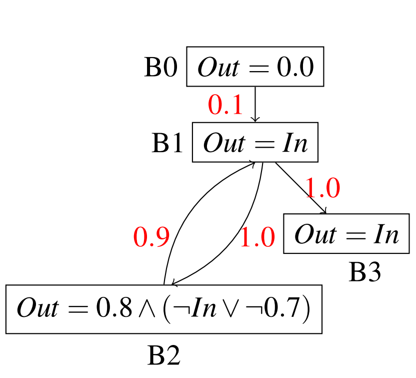
Our fuzzy data-flow analysis instead computes the partial truth of the property, i.e. values are elements of $[0,1]$. A value closer to 0 means that the property is biased towards false and vice versa. Furthermore the transfer functions are logical formulas from a Frank family fuzzy logic and the collector

[3]The general concept, allowing any $e \in [0,1]$, is called a *uninorm* [9] and is either orlike (i.e., $U(0,1) = U(1,0) = 1$) or andlike (i.e., $U(0,1) = U(1,0) = 0$). Our work does not require the full generality.

[4]This should not be confused with the partial order used in the interval abstraction.

functions are weighted average functions where the constant α is determined prior to performing the analysis. In contrast to the classical framework Kleene iteration proceeds until the results differ by a constant ε which is the maximal error allowed by the solution. The error can be made arbitrarily small.

This section introduces the fuzzy data-flow framework and we prove termination using continuity properties and Banach's fixed-point theorem. Section 4 then presents an example analysis to demonstrate the benefit of the framework. The analysis is performed on a weighted flow-graph $G = \langle V, E, \alpha \rangle$ where V is a set of logical formulas (denoting the transfer function of each block), $E \subseteq V \times V$ is a set of edges (denoting control transfers) and $\alpha_e \in [0,1]$ denotes the normalized contribution for each edge e. As a running example we will use Figure 1 (left) which shows a flow-graph with four nodes and their corresponding logical formula. The flow graph has four control edges denoting contributions between nodes. For instance, Block 1 (B1) receives 0.1 of its contribution from B0 and 0.9 from B2, i.e. $\alpha_{\langle B0,B1\rangle} = 0.1$ and $\alpha_{\langle B2,B1\rangle} = 0.9$.



$$\begin{cases} Out(B0) = 0.0 \\ Out(B1) = 0.1Out(B0) + 0.9Out(B2) \\ Out(B2) = \min(0.8, \max(1 - Out(B1), 0.3)) \\ Out(B3) = Out(B1) \end{cases}$$

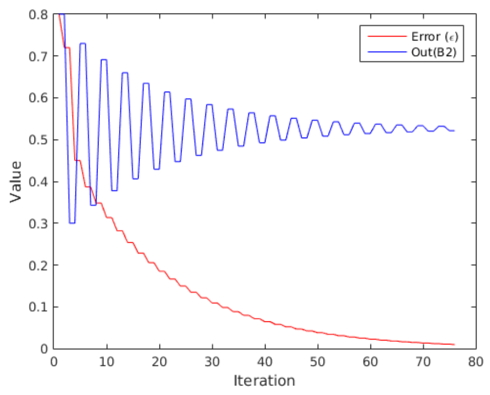


Figure 1: Example flow-graph (left) and its corresponding equation system (middle) and the analysis result and error as a function of iteration (right)

Definition 4. *Let $\mathscr{P}$ be a finite set of properties and $VS\colon \mathscr{P} \mapsto [0,1]$ a valuation for each property. We use $[\![\phi]\!](VS)$ to denote the interpretation of the fuzzy formula ϕ given a VS. Given a flow-graph $G = \langle V, E, \alpha \rangle$ with a unique start node v_{start} the map $GS\colon V \mapsto VS$ describes the value of each property at each node and a fuzzy data-flow analysis is a Kleene iteration of $F\colon GS \mapsto GS$:*

$$F(S) = \lambda v. \begin{cases} S(v_{start}) & v = v_{start} \\ \sum\limits_{\langle w,v\rangle \in E} \alpha_{\langle w,v\rangle} [\![v]\!](S(w)) & otherwise \end{cases}$$

Figure 1 (middle) shows the equation system, as implied by Definition 4, interpreted in a min-max fuzzy logic for the example flow-graph. The red colored text corresponds to the collector function, i.e. the weighted average, and the normal text is the interpretation of the logical formula. In order to prove termination of a fuzzy analysis we need to introduce a continuity property.

Definition 5. *A function $f\colon [0,1]^n \mapsto [0,1]$ is K-**Lipschitz continuous**[5] iff $\forall x,h\colon |f(\vec{x} - \vec{h}) - f(\vec{x})|_1 \le K|\vec{h}|_1$. Where $|\vec{x}|_1$ is l_1-norm (i.e., the absolute value) of $\vec{x}$[6]. If $0 \le K < 1$ then f is called a **contraction mapping** and if $0 \le K \le 1$ then f is called a **non-expansive mapping**.*

[5]Our definition restricts the domain and metric of both metric spaces (i.e., for the domain and co-domain of f) compared to the more general, and common, definition of a Lipschitz continuous function.

[6]Other l_p-norms can be used but only if we restrict the logic to the min-max fuzzy logic [15].

In a sequence of applications of a contraction mapping the difference between two consecutive applications will decrease and in the limit reach zero. By imposing a bounded error we guarantee that this sequence terminates in a bounded amount of time. The analysis error and result of B2 as a function of iteration for the example is shown in Figure 1 (right). Note that the error (red line) is decreasing and the value of B2 (blue line) tends towards a final value. We next proceed to prove that any fuzzy data-flow analysis iteratively computes more precise results and terminates in a bounded amount of time for a finite maximum error $\frac{1}{2^q}$ from some $q \in \mathbb{N} - \{0\}$. We let $[0,1]_q$ denote the maximal congruence set of elements from $[0,1]$ that are at least $\frac{1}{2^q}$ apart, i.e. $[0,1]_q = \{\frac{i}{2^q} \mid 0 \leq i \leq 2^q\}$. The set of intervals on $[0,1]$, i.e. $\mathbb{I}$ are defined analogously. For this we prove the *non-expansive* property of fuzzy formulas.

Theorem 1. *Let $x, y, C, w_i \in [0,1]_q$, for some $i \in \mathbb{N}$, $f_i(\vec{x})\colon [0,1]_q^n \mapsto [0,1]_q$ be 1-Lipschitz and $g_i(\vec{x})\colon [0,1]_q^n \mapsto [0,1]_q$ be K_i-Lipschitz.*

- *Functions $x+y$, $x-y$, xy, $\min(x,y)$ and $abs(x)$ are 1-Lipschitz. Constants are 0-Lipschitz.*
- *If $\sum_{i=0}^{N-1} w_i = 1$ then $\sum_{i=0}^{N-1} w_i f_i(\vec{x})$ is 1-Lipschitz.*
- *The composition $g_a \circ g_b$ is $K_a K_b$-Lipschitz.*

Finally,

- *Formulas defined in a Frank family Fuzzy logic are 1-Lipschitz.*
- *If $F\colon \mathbb{I}_q^n \to \mathbb{I}_q$ satisfies $\forall x \in \mathbb{I}_q^n\colon y \in x \Rightarrow f(y) \in F(x)$ then F is 1-Lipschitz.*

In summary, as per Theorem 1:

- Transfer functions in a Frank family fuzzy logic are non-expansive mappings.
- $S(v_{start})$ is constant and hence a contraction mapping.
- The composition of 1) Two non-expansive functions is a non-expansive function and 2) A non-expansive and a contraction function is a contraction function.

As the analysis is performed on the unit interval which together with the l_1-norm forms a complete metric space we can guarantee termination by Banach's fixed-point theorem.

Theorem 2 (Banach fixed-point theorem)**.** *Let (X,d) be a complete metric space and $f\colon X \mapsto X$ a contraction. Then f has a unique fixed-point x^* in X.*

This concludes our development of fuzzy data-flow analysis.

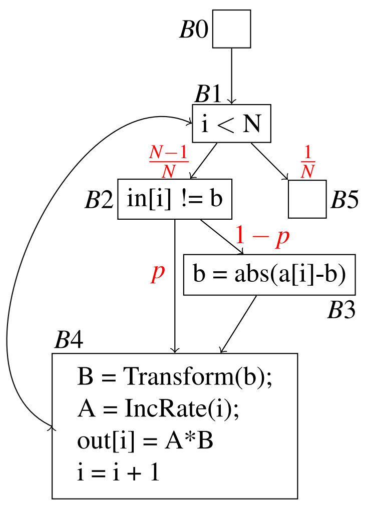
4 Lazy code motion

Improving performance often means removing redundant computations. Computations are said to be fully redundant, but not dead, if the operands at all points remain the same. For two such computations it is enough to keep one and store away the result for later. We can eliminate this redundancy using (global) common sub-expression elimination (GCSE). Furthermore a computation that does not change on some paths is said to be partially redundant. Loop invariant code motion (LICM) finds partially redundant computations inside loops and move these to the entry block of the loop. Lazy code motion is a compiler optimization that eliminate both fully and partially redundant computations, and hence subsumes both CSE and LICM. Knoop-Rüthing-Steffen (KRS) algorithm [13, 7] performs LCM in production compilers such as GCC when optimizing for speed. It consists of a series of data-flow analyses and can be summarized in these four steps:

1. Solve a very busy expression[7] and an available expression data-flow problem [17].
2. Introduce a set that describes the earliest block where an expression must be evaluated.

[7]Knoop et al. [13] refer to this as anticipatable expression data-flow problem.

```
void diffPCM() {
    b = 0, A = 0, B = 0;
    for(i = 0; i < N; i++)
        if(in[i] != b)
            b = abs(in[i]-b);
        B = Transform(b);
        A = IncRate(i);
        out[i] = A*B;
}
```



```
void diffPCM() {
    b = 0, A = 0, B = 0;
    for(i = 0; i < N; i++)
        Update← ANFIS decision of updating b
        Leave← ANFIS decision of leaving b
        if(in[i] != b)
            b = abs(in[i]-b);
            If Update < Leave: Decision error!
        else
            If Update > Leave: Decision error!
        B = Transform(b);
        A = IncRate(i);
        out[i] = A*B;
}
```

Figure 2: `diffPCM` function (left), the corresponding flow-chart (middle) and the version used in Section 4.3 which is annotated with ANFIS classifier invocations (right)

3. Determine the latest control flow edge where the expression must be computed.
4. Introduce *Insert* and *Delete* sets which describe where expressions should be evaluated.

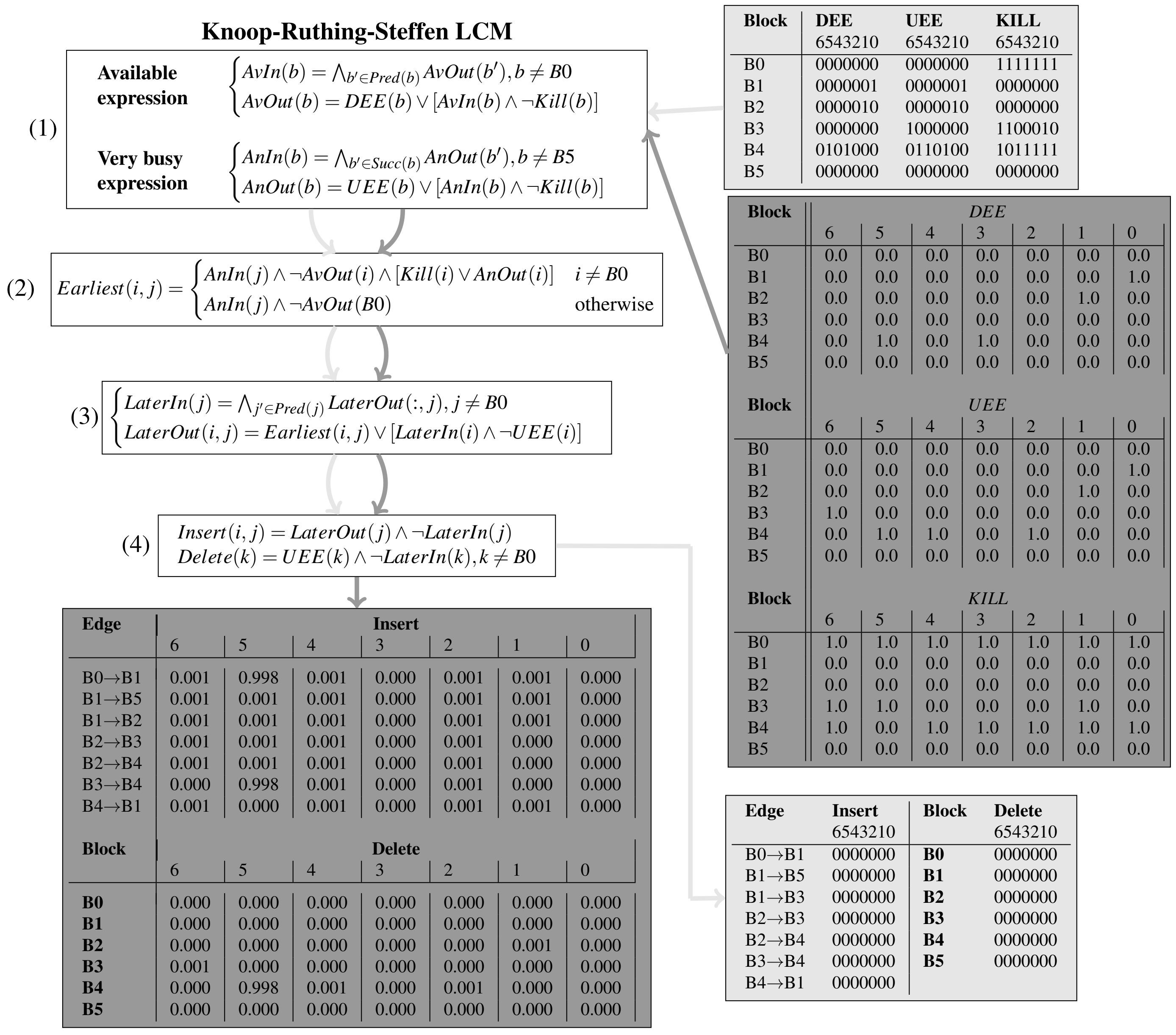
The target domain of the analysis is the set of static expressions in a program. Input to the analysis is three predicates determining properties about the expressions in different blocks:

- An expression "e" is *downward exposed* if it produces the same result if evaluated at the end of the block where it is defined. We use $DEE(b,e)$ to denote if "e" is downward exposed in block "b".
- An expression "e" is *upward exposed* if it produces the same result if evaluated at the start of the block where it is defined. We use $UEE(b,e)$ to denote this.
- An expression "e" is *killed* in block "b" if any variable appearing in "e" is updated in "b". We use $KILL(b,e)$ to denote this.

Very busy expression analysis is a backward-must data-flow analysis that depends on UEE and $KILL$ and computes the set of expressions that is guaranteed to be computed at some time in the future. Similarly Available expression analysis is a forward-must data-flow analysis that depends on DEE and $KILL$ and deduces the set of previously computed expressions that may be reused. The fixed-point system of these two analyses are shown in Figure 3. It is beyond the scope of this paper to further elaborate on the details of these analyses, the interested reader should consider Nielson et al. [17]. Here the LCM algorithm and the data-flow analyses it depends on, are applications we use to demonstrate the benefit of our framework. As such a rudimentary understanding is sufficient.

Consider the simplified differential pulse-code modulation routine `diffPCM` in Figure 2 (left). We assume that N and the relative number of times block $B3$ (denoted p) is statically known[8]. In each iteration `diffPCM` invokes the pure functions `Transform`, to encode the differential output, and `IncRate` to get a quantification rate. We use the KRS-LCM algorithm to determine if these invocations can be made prior to entering the loop and contrast this to a situation where the data-flow analyses are performed in the fuzzy framework. As we will show the "fuzzy KRS-LCM" allows us to uncover opportunites the classical KRS-LCM would miss.

[8]In this demonstration we let $p = 0.999$ and $N = 1000$, but our conclusions hold as N increases and p approaches 1.



Knoop-Ruthing-Steffen LCM

(1)

Available expression

$$\begin{cases} AvIn(b) = \bigwedge_{b' \in Pred(b)} AvOut(b'), b \neq B0 \\ AvOut(b) = DEE(b) \vee [AvIn(b) \wedge \neg Kill(b)] \end{cases}$$

Very busy expression

$$\begin{cases} AnIn(b) = \bigwedge_{b' \in Succ(b)} AnOut(b'), b \neq B5 \\ AnOut(b) = UEE(b) \vee [AnIn(b) \wedge \neg Kill(b)] \end{cases}$$

(2)

$$Earliest(i,j) = \begin{cases} AnIn(j) \wedge \neg AvOut(i) \wedge [Kill(i) \vee AnOut(i)] & i \neq B0 \\ AnIn(j) \wedge \neg AvOut(B0) & \text{otherwise} \end{cases}$$

(3)

$$\begin{cases} LaterIn(j) = \bigwedge_{j' \in Pred(j)} LaterOut(:, j), j \neq B0 \\ LaterOut(i,j) = Earliest(i,j) \vee [LaterIn(i) \wedge \neg UEE(i)] \end{cases}$$

(4)

$$Insert(i,j) = LaterOut(j) \wedge \neg LaterIn(j)$$
$$Delete(k) = UEE(k) \wedge \neg LaterIn(k), k \neq B0$$

Block	**DEE**	**UEE**	**KILL**
	6543210	6543210	6543210
B0	0000000	0000000	1111111
B1	0000001	0000001	0000000
B2	0000010	0000010	0000000
B3	0000000	1000000	1100010
B4	0101000	0110100	1011111
B5	0000000	0000000	0000000

Block	*DEE*						
	6	5	4	3	2	1	0
B0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
B2	0.0	0.0	0.0	0.0	0.0	1.0	0.0
B3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B4	0.0	1.0	0.0	1.0	0.0	0.0	0.0
B5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Block	*UEE*						
	6	5	4	3	2	1	0
B0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B1	0.0	0.0	0.0	0.0	0.0	0.0	1.0
B2	0.0	0.0	0.0	0.0	0.0	1.0	0.0
B3	1.0	0.0	0.0	0.0	0.0	0.0	0.0
B4	0.0	1.0	1.0	0.0	1.0	0.0	0.0
B5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Block	*KILL*						
	6	5	4	3	2	1	0
B0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
B1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B3	1.0	1.0	0.0	0.0	0.0	1.0	0.0
B4	1.0	0.0	1.0	1.0	1.0	1.0	1.0
B5	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Edge	**Insert**						
	6	5	4	3	2	1	0
B0→B1	0.001	0.998	0.001	0.000	0.001	0.001	0.000
B1→B5	0.001	0.001	0.001	0.000	0.001	0.001	0.000
B1→B2	0.001	0.001	0.001	0.000	0.001	0.001	0.000
B2→B3	0.001	0.001	0.001	0.000	0.001	0.000	0.000
B2→B4	0.001	0.001	0.001	0.000	0.001	0.000	0.000
B3→B4	0.000	0.998	0.001	0.000	0.001	0.000	0.000
B4→B1	0.001	0.000	0.001	0.000	0.001	0.001	0.000
Block	**Delete**						
	6	5	4	3	2	1	0
B0	0.000	0.000	0.000	0.000	0.000	0.000	0.000
B1	0.000	0.000	0.000	0.000	0.000	0.000	0.000
B2	0.000	0.000	0.000	0.000	0.000	0.001	0.000
B3	0.001	0.000	0.000	0.000	0.000	0.000	0.000
B4	0.000	0.998	0.001	0.000	0.001	0.000	0.000
B5	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Edge	**Insert**	**Block**	**Delete**
	6543210		6543210
B0→B1	0000000	**B0**	0000000
B1→B5	0000000	**B1**	0000000
B1→B3	0000000	**B2**	0000000
B2→B3	0000000	**B3**	0000000
B2→B4	0000000	**B4**	0000000
B3→B4	0000000	**B5**	0000000
B4→B1	0000000		

Expression	abs(a[i]-b)	Transform(b)	IncRate(i)	A*B	i+1	in[i] != b	$i \leq N$
Index	6	5	4	3	2	1	0

Figure 3: Knoop-Rüthing-Steffen LCM formulation (middle) using classical (left) and fuzzy (right/bottom) data-flow analysis

4.1 Type-1 static analysis

The data-flow problems of the KRS algorithm use expressions as domain. The mapping between expressions of `diffPCM` and indexes are listed in Figure 3 (bottom) together with the values of DEE, UEE and $KILL$ for each block (top right). The classical KRS algorithm conclude that both calls must be evaluated in B4 (bottom light gray box, "Delete" matrix, Column 4 and 5).

For the fuzzy data-flow analyses we use the Type-1 Min-Max fuzzy logic. The corresponding fuzzy sets of DEE, UEE and $KILL$ are given in Figure 3 (top dark gray box). Step (1) of the fuzzy KRS-LCM is hence the fixed-point to below system of equations:

Available expression analysis system

$$\begin{cases} AvOut(B0) = 0.0 \\ AvOut(B1) = DEE(B1) \vee \left(\left[\frac{1}{N}AvOut(B0) + \frac{N-1}{N}AvOut(B4)\right] \wedge \neg Kill(B1)\right) \\ AvOut(B2) = DEE(B2) \vee (AvOut(B1) \wedge \neg Kill(B2)) \\ AvOut(B3) = DEE(B3) \vee (AvOut(B2) \wedge \neg Kill(B3)) \\ AvOut(B4) = DEE(B4) \vee ([pAvOut(B2) + (1-p)AvOut(B3)] \wedge \neg Kill(B4)) \\ AvOut(B5) = DEE(B5) \vee (AvOut(B1) \wedge \neg Kill(B4)) \end{cases}$$

Very busy expression analysis system

$$\begin{cases} AnOut(B0) = UEE(B0) \vee (AnOut(B1) \wedge \neg Kill(B1)) \\ AnOut(B1) = UEE(B1) \vee \left(\left[\frac{N-1}{N}AnOut(B2) + \frac{1}{N}AnOut(B5)\right] \wedge \neg Kill(B1)\right) \\ AnOut(B2) = UEE(B2) \vee ([pAnOut(B4) + (1-p)AnOut(B3)] \wedge \neg Kill(B2)) \\ AnOut(B3) = UEE(B3) \vee (AnOut(B4) \wedge \neg Kill(B3)) \\ AnOut(B4) = UEE(B4) \vee (AnOut(B1) \wedge \neg Kill(B4)) \\ AnOut(B5) = 0.0 \end{cases}$$

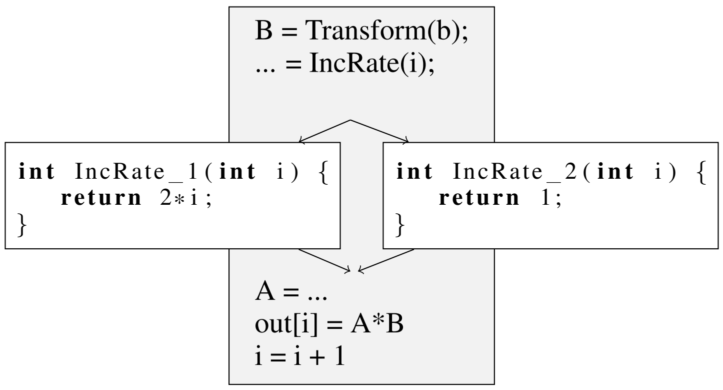
Steps (2) and (4) introduce (constant) predicates and are performed outside the analysis framework. Step (3) is done similarly to step (1). Figure 3 (bottom dark gray box) shows the result from step (4). In contrast to the classical LCM the result implies that it is *very plausible* (0.998) that we can delete the invocation of `Transform` ("Delete" matrix, Column 5) from block B4 and instead add it at the end of B0 and B3 (or start of B1 and B4). However, result for the invocation of `IncRate` remains. This is because the invocation depends on the value of i which is updated at the end of B4.

4.2 Type-2 static analysis

To increase data-flow analysis precision a function call is sometimes inlined at the call site. The improvement can however be reduced if the control-flow analysis is inaccurate and multiple targets are considered for a particular call site. We show how the uncertainty in control-flow and data-flow can be quantified in two different dimensions using type-2 interval fuzzy sets. As per Section 2 we can lift an arbitrary fuzzy predicate to intervals. Here we assume no knowledge about the relative number of calls to each target and treat the different calls non-deterministically.

We assume two different `IncRate` functions, as in Figure 4 (left), have been determined as targets. Their respective UEE and $Kill$ entries are the same but since i is updated at the end of block B4 their DEE entry will differ. The result of `IncRate_1` depends on the variable i and therefore $DEE(B4_1) = \mathbf{0101000}$, in contrast the entry for `IncRate_2` is $DEE(B4_2) = \mathbf{0111000}$, where $\mathbf{0} = [0,0]$ and $\mathbf{1} = [1,1]$. The new entry for block B4 is given by $DEE(B4) = DEE(B4_1)\,\tilde{\vee}\,DEE(B4_2) = \langle \mathbf{0}, \mathbf{1}, [0,1], \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0} \rangle$. The new $Kill$, DEE and UEE sets are given in Figure 4 (right).

Applying the fuzzy KRS-LCM, but with Type-1 min-max fuzzy logic lifted to Interval type-2 min-max fuzzy logic gives the values of *Delete* and *Insert* for expression `IncRate(i)` in Figure 4 (right).



```
B = Transform(b);
... = IncRate(i);
```

```
int IncRate_1(int i) {
    return 2*i;
}
```

```
int IncRate_2(int i) {
    return 1;
}
```

```
A = ...
out[i] = A*B
i = i + 1
```

Block	Kill	DEE	UEE
6	$[1.0, 1.0]$	$[0.0, 0.0]$	$[0.0, 0.0]$
5	$[0.0, 0.0]$	$[1.0, 1.0]$	$[1.0, 1.0]$
4	$[1.0, 1.0]$	$[0.0, 1.0]$	$[1.0, 1.0]$
3	$[1.0, 1.0]$	$[1.0, 1.0]$	$[0.0, 0.0]$
2	$[1.0, 1.0]$	$[0.0, 0.0]$	$[1.0, 1.0]$
1	$[1.0, 1.0]$	$[0.0, 0.0]$	$[0.0, 0.0]$
0	$[1.0, 1.0]$	$[0.0, 0.0]$	$[0.0, 0.0]$

Edge	Insert	Block	Delete
B0→B1	$[0.001, 0.999]$	B0	$[0.000, 0.000]$
B1→B5	$[0.001, 0.999]$	B1	$[0.000, 0.000]$
B1→B3	$[0.001, 0.999]$	B2	$[0.000, 0.000]$
B2→B3	$[0.001, 0.999]$	B3	$[0.000, 0.000]$
B2→B4	$[0.001, 0.999]$	B4	$[0.002, 0.999]$
B3→B4	$[0.001, 0.999]$	B5	$[0.000, 0.000]$
B4→B1	$[0.000, 0.999]$		

Figure 4: Implementations of `IncRate` inlined in block B4 (left); DEE, UEE and Kill vectors of block B4 and *Delete Insert* analysis result for expression `IncRate(i)` (right)

The result for invoking `IncRate` prior to the loop is $[0.001, 0.999]$ as opposed to 0.001 from the Type-1 analysis in Section 4.1. The added dimension in the result of the type-2 fuzzy analysis allows us to differentiate uncertain results from pessimistic results. In the given example we showed that the result of Section 4.1 is a pessimistic over-generalization and that the two paths need to be considered seperately to increase precision.

4.3 Hybrid analysis

The result from a fuzzy data-flow analysis is a set of fuzzy membership degrees. This section shows how the result can automatically be improved following the static analysis using a fuzzy regulator/classifier, if more specific information is provided at a later point. The classifier, a Takagi-Sugeno Adaptive-Network-based fuzzy inference system (TS-ANFIS) [11, 12] shown in Figure 5, is composed of five layers:

1. Lookup fuzzy membership degree of the input value.
2. Compute the *firing strength of a rule*, i.e. conjunction of all membership degrees from each rule.
3. Normalize the firing strengths, i.e., $\bar{w}_i = w_i / \sum_j w_j$.
4. Weight the normalized firing strength to the consequent output of the rule $f_i(x)$.
5. Combine all rule classifiers, i.e. $f = \sum_i \bar{w}_i f_i$.

This classifier uses a polynomial (i.e., the consequent part of the adaptive IF-THEN rules) to decide the output membership. The order of the TS-ANFIS is the order of the polynomial. The classification accuracy of the TS-ANFIS can be improved online/offline by fitting the polynomial to the input data. For a first-order TS-ANFIS this can be implemented as follows:

- (*Offline*) (Affine) Least square (LS) optimization [11] is a convex optimization problem that finds an affine function (i.e., $y = a_0 + \sum_{i=1}^{n} a_i x_i$) which minimizes $||A[1;X] - Y||_2^2$ where X and Y are the input and output vectors of the training set.
- (*Online*) Least mean square (LMS) [11] is an adaptive filter that gradually (in steps of a given constant μ) minimizes $\mathbb{E}\left[|y - f(x)|^2\right]$, where $\langle x, y \rangle$ is an input/output sample.

To exemplify the functionality of the TS-ANFIS we consider the classification of $\vec{x} = \langle 0.6, 0.2 \rangle$ using the two rule TS-ANFIS from Figure 5 (left). Let $f_1(\vec{x}) = 0.2x_0 - 0.43x_1$, $f_2(\vec{x}) = 0.1x_1 + 0.5$ and membership functions be given as in Figure 5 (right). The membership degrees are marked in the figure as $\mu_{A0}(x0) =$

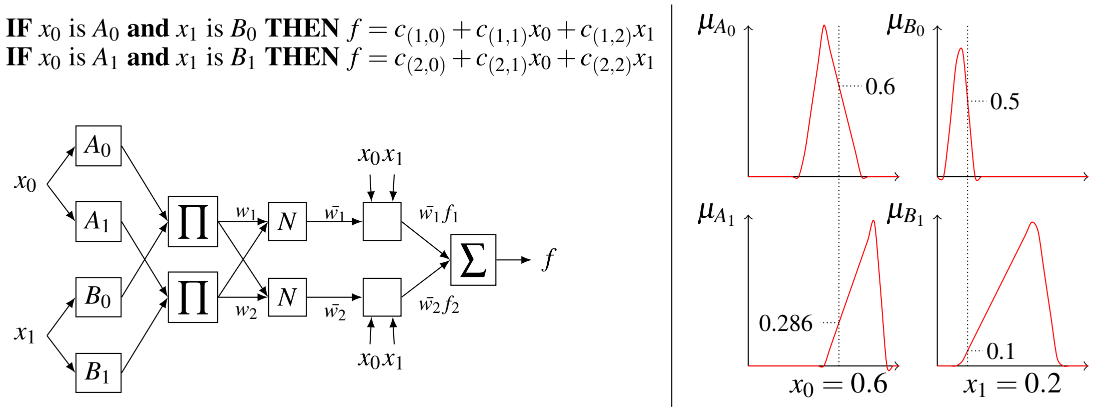


Figure 5: First-order Takagi-Sugeno ANFIS with two rules and two variables (left) and four example fuzzy sets (right)

0.6, $\mu_{B0}(x1) = 0.5$ for the first rule and $\mu_{A1}(x0) = 0.286$, $\mu_{B1}(x0) = 0.1$ for the second rule. Hence the weight of the first rule (i.e., w_1) is $0.6 \wedge 0.5 = 0.5$ and the second rule (i.e., w_2) is $0.286 \wedge 0.1 = 0.1$. The normalized weights are then $\bar{w_1} = 0.833$ and $\bar{w_1} = 0.167$. As the consequence functions output $f_1(\vec{x}) = 0.034$ and $f_2(\vec{x}) = 0.52$ we produce the prediction $0.833 f_1(\vec{x}) + 0.167 f_2(\vec{x}) = 0.115$.

We return to the `diffPCM` function and again consider if we can invoke `Transform(b)` prior to entering the loop. We saw in Section 4.1 that the fuzzy membership degree was 0.998. To improve classification accuracy we let the TS-ANFIS also use the i variable and the first input value (i.e., $in[0]$). These variables were not part of the analysis and so we conservatively assume the fuzzy membership degree to be the same for any value of these variables (in our experiments: 1.0). As shown in Figure 2 (right), we inserted calls to compute the ANFIS decision of updating and keeping the variable b constant in the `diffPCM` function. If the incorrect decision was made the error was noted and an error rate computed after handling all input samples.

We consider invoking the `diffPCM` function on four different input sets. Each input set defined as 10 periods with 25 input values in each period. The input sets (i.e., `in[...]`) is given in Figure 6 (top). We use the LMS algorithm[9] after each incorrect classification and the LS algorithm if the error rate of a period was larger than or equal to 80%. Note that the values of a period is not always perfectly representable by a linear classifier and sometimes varies between different periods, although periods are "*similar*". Hence we do not expect the classifier to be monotonically improving with increasing period. As shown in the result in Figure 6 (bottom) the classification error decreases fast with both period and input sample. In two cases a small residual error remains after the final period. This show that the TS-ANFIS can improve the analysis result dynamically and hence increase the accuracy of when `Transform` can be invoked prior to entering the loop.

5 Related work

Most systems include elements (e.g., input values, environment state) where information is limited but probabilistic and/or non-deterministic uncertainty can be formulated. For these systems a *most likely* or even *quantitative analysis* of properties is possible. Often this analysis relies on a probability theory for

[9]The constant μ for the four different runs was set to 0.001, 0.05, 0.15 and 0.1 respectively.

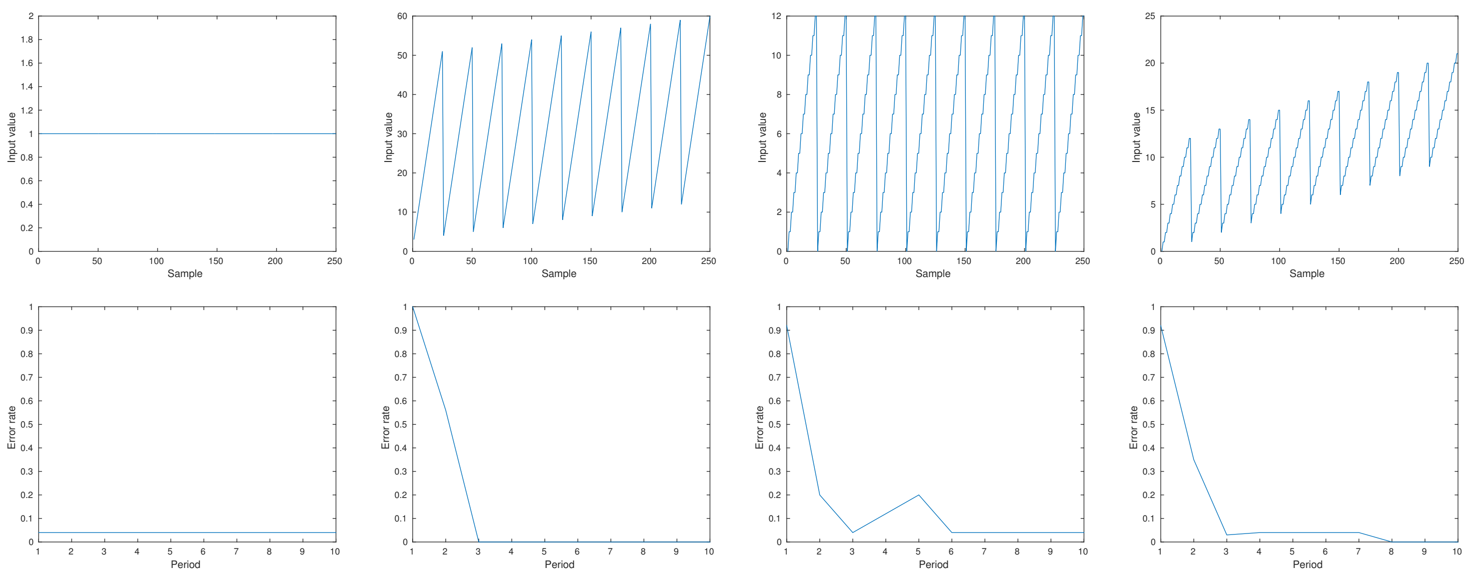


Figure 6: 10×25 input values (top) and the corresponding classification error rate (bottom)

logical soundness. Cousot and Monerau [3] introduced a unifying framework for probabilistic abstract interpretation. Much work have since, although perhaps implicitly, relied on their formulation. Often probabilistic descriptions are known with imprecision that manifests as non-deterministic uncertainty [2]. Adje et al. [1] introduced an abstraction based on the zonotope abstraction for Dempster-Shafer structures and P-boxes[10].

Di Pierro et al. [6] developed a probabilistic abstract interpretation framework and demonstrated an alias analysis algorithm that could guide the compiler in this decision. They later formulated data-flow problems (e.g., liveness analysis) in the same framework [5]. An important distinction between their (or similar probabilistic frameworks) and classical frameworks is the definition of the confluence operator. In contrast to a classical may- or must framework they use the weighted average. This is similar to the work by Ramalingam [20] that showed that the meet-over-paths (MOP) solution exists for such confluence operator with a transfer function defined in terms of min, max and negation (i.e., the Min-max fuzzy logic). Our work extends this to allow other transfer functions and integrates the static data-flow analysis with a dynamic refinement mechanism through fuzzy control theory.

6 Conclusion

A major problem for static program analysis is the limited input information and hence the conservative results. To alleviate the situation dynamic program analysis is sometimes used. Here accurate information is available, but in contrast to its static counter-part the results only cover a single or few runs. To bridge the gap, and find a promising middle-ground, probabilistic/speculative program analysis frameworks have been proposed. These frameworks can be considered to intersect both by being a static program analysis that uses dynamic information. We have introduced an abstraction based on fuzzy sets that supports such analyses. We applied our abstraction to data-flow problems of use for speculative compilation and showed how our analysis unveils opportunities that previous approaches could

[10]Lower and upper bounds on a cumulative probability distribution functions

not express and reason about. We furthermore showed that an abstraction based on fuzzy sets admit mechanisms from fuzzy control theory to enhance the analysis result dynamically allowing for a hybrid analysis framework.

References

[1] Assale Adje, Olivier Bouissou, Jean Goubault-Larrecq, Eric Goubault & Sylvie Putot (2014): *Static Analysis of Programs with Imprecise Probabilistic Inputs*. In: *Verified Software: Theories, Tools, Experiments*, *Lecture Notes in Computer Science* 8164, Springer Berlin Heidelberg, pp. 22–47.

[2] Patrick Cousot & Radhia Cousot (2014): *Abstract Interpretation: Past, Present and Future*. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, ACM, pp. 2:1–2:10.

[3] Patrick Cousot & Michaël Monerau (2012): *Probabilistic Abstract Interpretation*. In: *22nd European Symposium on Programming (ESOP 2012)*, *Lecture Notes in Computer Science* 7211, Springer-Verlag, pp. 166–190.

[4] Jeff Da Silva & J. Gregory Steffan (2006): *A Probabilistic Pointer Analysis for Speculative Optimizations*. In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, ACM, pp. 416–425.

[5] Pierro A Di & H Wiklicky (2013): *Probabilistic data flow analysis: a linear equational approach*. In: *Proceedings of the Fourth International Symposium on Games, Automata, Logics and Formal Verification*, pp. 150–165.

[6] Alessandra Di Pierro, Chris Hankin & Herbert Wiklicky (2007): *A Systematic Approach to Probabilistic Pointer Analysis*. In: *Programming Languages and Systems*, *Lecture Notes in Computer Science* 4807, Springer Berlin Heidelberg, pp. 335–350.

[7] Karl-Heinz Drechsler & Manfred P. Stadel (1993): *A Variation of Knoop, RüThing, and Steffen's Lazy Code Motion*. *SIGPLAN Not.* 28(5), pp. 29–38.

[8] D. Dubois & H. Prade (1980): *Fuzzy sets and systems - Theory and applications*. Academic press, New York.

[9] D. Dubois, H.M. Prade & H. Prade (2000): *Fundamentals of Fuzzy Sets*. The Handbooks of Fuzzy Sets, Springer US.

[10] Mai Gehrke, Carol Walker & Elbert Walker (1996): *Some comments on interval valued fuzzy sets*. *International Journal of Intelligent Systems* 11(10), pp. 751–759.

[11] J.-S.R. Jang (1993): *ANFIS: adaptive-network-based fuzzy inference system*. *Systems, Man and Cybernetics, IEEE Transactions on* 23(3), pp. 665–685.

[12] Jyh-Shing Roger Jang & Chuen-Tsai Sun (1997): *Neuro-fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[13] Jens Knoop, Oliver Rüthing & Bernhard Steffen (1992): *Lazy Code Motion*. In: *Proceedings of the ACM SIGPLAN 1992 Conference on Programming Language Design and Implementation*, PLDI '92, ACM, pp. 224–234.

[14] S. Maleki, Yaoqing Gao, M.J. Garzaran, T. Wong & D.A. Padua (2011): *An Evaluation of Vectorizing Compilers*. In: *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pp. 372–382.

[15] A. Mesiarov (2007): *k-lp-Lipschitz t-norms*. *International Journal of Approximate Reasoning* 46(3), pp. 596 – 604. Special Section: Aggregation Operators.

[16] Markus Mock, Manuvir Das, Craig Chambers & Susan J. Eggers (2001): *Dynamic Points-to Sets: A Comparison with Static Analyses and Potential Applications in Program Understanding and Optimization*. In:

Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE '01, ACM, pp. 66–72.

[17] Flemming Nielson, Hanne R. Nielson & Chris Hankin (1999): *Principles of Program Analysis*. Springer-Verlag New York, Inc.

[18] P.M. Petersen & D.A. Padua (1996): *Static and dynamic evaluation of data dependence analysis techniques*. *Parallel and Distributed Systems, IEEE Transactions on* 7(11), pp. 1121–1132.

[19] Dimitrios Prountzos, Roman Manevich, Keshav Pingali & Kathryn S. McKinley (2011): *A Shape Analysis for Optimizing Parallel Graph Programs*. In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, ACM, pp. 159–172.

[20] G. Ramalingam (1996): *Data Flow Frequency Analysis*. In: *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, PLDI '96, ACM, pp. 267–277.

[21] ConstantinoG. Ribeiro & Marcelo Cintra (2007): *Quantifying Uncertainty in Points-To Relations*. In: *Languages and Compilers for Parallel Computing*, *Lecture Notes in Computer Science* 4382, Springer Berlin Heidelberg, pp. 190–204.

Appendix A: Omitted proofs

of Theorem 1. Let $x,y,C,w_i \in [0,1]$, for some $i \in \mathbb{N}$, $f_i(\vec{x}) : [0,1]_q^n \mapsto [0,1]_q$ be 1-Lipschitz and $g_i(\vec{x}) : [0,1]_q^n \mapsto [0,1]_q$ be K_i-Lipschitz.

1. *Functions $x+y$, $x-y$, xy, $\min(x,y)$ and $abs(x)$ are 1-Lipschitz. Constants are 0-Lipschitz* Let $b \in [x, x+h]$ for some $0 \le x \le x+h \le 1$:

   (a) $g(x) = abs(x)$:

$$\begin{aligned} |g(x+h)-g(x)| &= |abs(x+h)-abs(x)| && \text{By definition} \\ &= |x+h-x| && x,h \in [0,1] \\ &\le 1|h| \end{aligned}$$

   (b) $g(x,y) = x+y$:

$$\begin{aligned} |g(x+h_1,y+h_2)-g(x,y)| &= |((x+h_1)+(y+h_2))-(x+y)| && \text{By definition} \\ &= |h_1+h_2| \\ &\le |h_1|+|h_2| && \text{Triangle inequality} \\ &= 1|h| && \text{Distributivity} \end{aligned}$$

   (c) $g(x,y) = x-y$:

$$\begin{aligned} |g(x+h_1,y+h_2)-g(x,y)| &= |((x+h_1)-(y+h_2))-(x-y)| && \text{By definition} \\ &= |h_1+(-1)h_2| \\ &\le |h_1|+|-1||h_2| && \text{Triangle inequality} \\ &= 1|h| && \text{Distributivity} \end{aligned}$$

   (d) $g(x,y) = xy$:

$$\begin{aligned} |g(x+h_1,y+h_2)-g(x,y)| &= |((x+h_1)(y+h_2))-(xy)| && \text{By definition} \\ &= |h_1y+xh_2| \\ &\le |h_1+h_2| && 0 \le x,y \le 1 \\ &\le |h_1|+|h_2| && \text{Triangle inequality} \\ &= 1|h| && \text{Distributivity} \end{aligned}$$

(e) $g(x,y) = \min(x,y)$:

$$\begin{aligned}
|g(x+h_1,y+h_2)-g(x,y)| &= |\min(x+h_1,y+h_2)-\min(x,y)| && \text{By definition} \\
&= \left|\left(\frac{x+h_1+y+h_2}{2}-\frac{|x+h_1-y-h_2|}{2}\right) - \left(\frac{x+y}{2}-\frac{|x-y|}{2}\right)\right| && min(x,y)=\frac{x+y}{2}-\frac{|x-y|}{2} \\
&\le \left|\frac{h_1+h_2}{2}-\frac{|x|+|-1||y|+|h_1-h_2|}{2} + \frac{|x|+|-1||y|}{2}\right| \\
&= \left|\frac{h_1+h_2}{2}-\frac{|h_1-h_2|}{2}\right| \\
&= |\min(h_1,h_2)| \\
&\le |h_1+h_2| \\
&= 1|h| && \text{Triangle inequality}
\end{aligned}$$

(f) $g(x,y) = C$

$$|g(x+h)-g(x)| = |C-C| = 0 \le 0|h|$$

2. *If* $\sum_{i=0}^{N-1} w_i = 1$ *then* $\sum_{i=0}^{N-1} w_i f_i(\vec{x})$ *is* 1*-Lipschitz*

$$\begin{aligned}
\left|g(\vec{x}+\vec{h})-g(\vec{x})\right| &= \left|\sum_{i=0}^{N-1} w_i f(x\vec{+}h) - \sum_{i=0}^{N-1} w_i f(\vec{x})\right| && \text{By definition} \\
&= \left|\sum_{i=0}^{N-1} w_i\left(f(x\vec{+}h)-f(\vec{x})\right)\right| && \text{Associativitiy and commutativity} \\
&\le \left(\sum_{i=0}^{N-1} w_i K_i\right)|h| && \text{Triangle inequality, distributivity and } w_i \ge 0 \\
&= 1|h| && K_i = 1 \text{ and } \sum_i w_i = 1
\end{aligned}$$

3. *The composition* $g_a \circ g_b$ *is* K_aK_b*-Lipschitz* $g(\vec{x}) = f_a(\vec{x}) \circ f_a(\vec{x})$:

$$\begin{aligned}
|g(x+h)-g(x)| &= \left|f_a(f_b(x\vec{+}h)) - f_a(f_b(\vec{x}))\right| && \text{By definition} \\
&\le K_a\left|f_b(x\vec{+}h)-f_b(\vec{x})\right| && \text{Definition 5} \\
&\le K_aK_b|h| && \text{Definition 5}
\end{aligned}$$

4. *Formulas defined in a Frank family Fuzzy logic are* 1*-Lipschitz.* This follows from structural induction on the height of parse tree of the predicate $P(x)$. By De Morgan's laws it is enough to show the induction step for $\vee$ and $\neg$.
   - **Base case**:
     - v: $g(x) = x$: $|g(x+h)-g(x)| = |x+h-x| \le 1|h|$.
     - $\top$ or $\bot$: $g(x) = C$: 1-Lipschitz by 3.

- **Induction step**:
  - $\neg\phi \equiv 1-\phi$: 1-Lipschitz from the base case for constants ($\top$ and $\bot$) and cases 1c and 3 and assumption that ϕ is 1-Lipschitz.
  - $\phi_1 \vee \phi_2$: $\begin{cases} \min(x,y) & \text{1-Lipschitz from Theorem 1 case 1e} \\ xy & \text{1-Lipschitz from Theorem 1 case 1d} \\ \max(x+y-1,0) & \text{Equal to } 1-\min(2-x-y,1) \text{ using De Morgans law.} \\ & \text{This expression is 1-Lipschitz from Theorem 1 case 1c, 1e and 3} \end{cases}$

5. *If $F : \mathbb{I}_q^n \to \mathbb{I}_q$ satisfy $\forall x \in \mathbb{I}_q^n : y \in x \Rightarrow f(y) \in F(x)$ then F is 1-Lipschitz*

   $F : \mathbb{I}_q^n \to \mathbb{I}_q$ can be decomposed into two functions $F_l : \mathbb{I}_q^n \to [0,1]_q$ and $F_u : \mathbb{I}_q^n \to [0,1]_q$ such that $\forall i : F(i) = [F_l(i), F_u(i)]$, i.e., $F_l(i)$ gives the infimum of $f(i)$ and $F_u(i)$ gives the supremum of i. We show that both F_l and F_u are 1-Lipschitz continuous:

   - $F_l(I) = \inf_{i\in I} f(i)$: Assume $I = [l,u]$, since I is finite we can rewrite the operation as pair-wise applications of min, i.e., $min(f(l), min(f(l+\frac{1}{2^q}), min(f(l+\frac{2}{2^2}), ...)))$. As per above case 1e min is 1-Lipschitz. Similarly the composition of two 1-Lipschitz functions is also 1-Lipschitz, or in extension, a finite number of compositions.
   - $F_u(I) = \sup_{i\in I} f(i)$: $\max(x,y)$ is equivalent to $1-\min(1-x, 1-y)$ which is 1-Lipschitz by above case 1c and 1e so proof follows in the same way as the $F_l(I)$ case.

□