Generic Abstract Domains

- A generic abstract domain is parameterized.
- A particular abstract domain instantiation: bind the formal parameters to program dependent actual parameters (constants, variables, control points, etc.)
- Example: Kildall [9]'s generic abstract domain for constant propagation $D(C, V)$ is:
  \[
  D(C, V) = \bigcap_{t \in C} \bigcup_{X \subseteq V(t)} L. 
  \]
  - $L$ is Kildall’s complete lattice. Given a command $C$, it is instantiated to $D(\text{lab}[C], \text{var}[C])$ where
    - $\text{lab}[C]$ is the set of labels of command $C$
    - $\text{var}[C](\ell)$ is the set of program variables $X$ which are visible at this program point $\ell$ of command $C$.

Generic Comparison Abstract Domain

We let $D_{\text{rel}}(X)$ be a generic relational integer abstract domain parameterized by a set $X$ of program and auxiliary variables (such as octagons [12, 13] or polyhedra [7]). This abstract domain is assumed to have abstract operations on $r_1, r_1, r_2 \in D_{\text{rel}}(X)$ such as:

- the projection or variable elimination $\exists x \in X : r$,  
- disjunction $r_1 \lor r_2$,  
- conjunction $r_1 \land r_2$,  
- abstract predicate transformers for assignments and tests, etc.
Then we define the generic comparison abstract domain:

\[ D_{lt}(X) = \{ (lt(t, a, b, c, d), r) \mid t \in X \land a, b, c, d \notin X \land r \in D_{rel}(X \cup \{a, b, c, d\}) \} . \]

The meaning of \( (lt(t, a, b, c, d), r) \) of an abstract predicate \( (lt(t, a, b, c, d), r) \)
is informally that all elements of \( t \) between indices \( a \) and \( b \) are less than any element of \( t \) between indices \( c \) and \( d \) and moreover \( r \) holds:

\[
\gamma((lt(t, a, b, c, d), r)) = \exists a, b, c, d : \begin{align*}
  t.l \leq a & \leq b \leq t.h \\
  \land \ t.l \leq c & \leq d \leq t.h \\
  \land \forall i \in [a, b] : \forall j \in [c, d] : t[i] \leq t[j] & \land r
\end{align*}
\]

where \( t.l \) is the lower bound and \( t.h \) is the upper bound of the indices \( i \) of the array \( t \) with elements \( t[i] \).
**Abstract Implication**

We have \(\text{lt}(t, a, b, c, d, r) \Rightarrow r\). If \(r \Rightarrow r'\) and \(a \leq b \leq c \leq d\) and \(e \leq f \leq g \leq h\) then:

\[
\text{lt}(t, a, d, e, h, r) \Rightarrow \text{lt}(t, b, c, f, g), r'
\]

as shown below:

![Diagram](image1)

\((1)\)

**Abstract Conjunction (Cont'd)**

If \(a \leq b \leq c \leq d\) and \(e \leq f \leq g \leq h\) then we have:

\[
\text{lt}(t, a, c, f, h), r) \land \text{lt}(t, b, d, e, g), r'
\]

\[
= \text{lt}(t, b, c, f, g), \exists a, d, e, h : r \land r'
\]

as shown below:

![Diagram](image2)

\((2)\)

**Abstract Conjunction (End)**

The same way:

\[
\text{lt}(t, a, b, c, e, r) \land \text{lt}(t, d, f, g, h), r'
\]

\[
= \text{lt}(t, a, b, g, h), \exists c, e, d, f : r \land r'
\]

we have:

\[
\text{lt}(t, a, b, c, e, r) \land \text{lt}(t, d, f, g, h), r'
\]

\[
= \text{lt}(t, a, b, g, h), \exists c, e, d, f : r \land r'
\]

\((3)\)

when \((r \land r') \Rightarrow (c \leq d \leq e \leq f)\).
Abstract Disjunction

We have:
\[
\text{lt}(t, a, b, c, d), r) \lor \text{lt}(t, e, f, g, h), r' =
\]
\[
(\exists a, b, c, d : i = a, j = b, k = c, l = d \land r) \lor
\]
\[
(\exists e, f, g, h : i = e, j = f, k = g, l = h \land r')
\]

(4)

Abstract Predicate Transformers for the Generic Comparison Abstract Domain

- Then the abstract domain must be equipped with abstract predicate transformers for tests, assignments, etc.
- We consider forward strongest postconditions (although weakest preconditions, which avoid an existential quantifier in assignments, may sometimes be simpler [14]).
- We depart from traditional predicate abstraction which uses a simplifier (or a theorem prover) to formally evaluate the abstract predicate transformer \( \alpha \circ F \circ \gamma \) approximating the concrete predicate transformer \( F \).

Abstract Disjunction (cont’d)

In case one of the terms does not refer to the array (\( t \notin \text{var}[r] \)), a criterion must be used to force the introduction of an identically true array term \( \text{lt}(t, i, i, i, i) \). For example if the auxiliary variables \( d, f, g, h \) in \( r' \) depend upon one selectively chosen variable \( I \), then we have:

\[
r \lor \text{lt}(t, d, f, g, h), r') =
\]
\[
(\text{lt}(t, i, j, k, l), (i = j = k = l = I) \land r) \lor
\]
\[
(\exists d, f, g, h : i = d \land j = f \land k = g \land l = h \land r')
\]

(5)

This case appears typically in loops, which can also be handled by unrolling, see 3.1.

- The alternative proposed below is traditional in static program analysis and directly provides an over-approximation of the best abstract predicate transformer \( \alpha \circ F \circ \gamma \) in the form of an algorithm (which correctness must be established formally).
- The simplifier/prover/pattern-matcher is used only to reduce the post-condition in the normal form (??) which is required for the abstract predicates.
Abstract Strongest Postconditions for Tests

\{ P_1 \}

\textbf{if} \ (t[I] > t[I+1]) \ \textbf{then}
\{ P_1 \land \langle \text{lt}(t, i, j, k, \ell), i = I \land j = I + 1 \land k = I \land \ell = I \rangle \} \quad (7)
\overbrace{\ldots}^{(9)}
\{ P_2 \}

\textbf{else}
\{ P_1 \land \langle \text{lt}(t, i, j, k, \ell), i = I \land j = k = \ell = I + 1 \rangle \} \quad (8)
\overbrace{\ldots}^{(9)}
\{ P_3 \}

\textbf{fi}
\{ P_2 \lor P_3 \} \quad (9)

Abstract Strongest Postconditions for Assignments (Cont’d)

The same way if \( t \not\in \text{var}[r] \) and \( r = (I \in [i, j] \land J \in [i, J]) \lor (J \in [k, \ell] \land I \in [k, \ell]) \) then:
\{ \langle \text{lt}(t, i, j, k, \ell), r \rangle \}
t[I] := t[I]
\{ \langle \text{lt}(t, i, j, k, \ell), r \rangle \} \quad (11)

since the swap of the array elements does not interfere with the assertions.

Generic Comparison Widening

Finally the abstract domain must be equipped with a widening (and optionally a narrowing to improve precision) to speed up the convergence of iterative fixpoint computations [4]. We choose to define the widening \( \triangledown \) as:
\[ \langle \text{lt}(t, i, j, k, \ell), r \rangle \triangledown \langle \text{lt}(t, m, n, p, q), r' \rangle = \langle \text{lt}(t, i, j, k, \ell), r \rangle \lor \langle \text{lt}(t, m, n, p, q), r' \rangle \] in
\[ \langle \text{lt}(t, r, s, t, u), r'' \rangle. \]

For assignment, assuming \( t \not\in \text{var}[r] \) and \( r = (I = I \land j = I + 1 \land k = I \land \ell = I) \), we have:
\[ \{ \langle \text{lt}(t, i, j, k, \ell), r \rangle \} \]
\[ t[I] := t[I+1] \]
\{ \langle \text{lt}(t, m, n, p, q), \exists i, j, k, \ell : r \land m = I \land n = p = q = I + 1 \rangle \}. \quad (10) \]
Generic Comparison Widening (Cont’d)

Typically, when handling loops, one encounters widenings of the form \( r \lor \langle \text{lt}(t, m, n, p, q), r' \rangle \) where \( r \) corresponds to the loop entry condition while the term \( \text{lt}(t, m, n, p, q) \) appears during the analysis of the loop body. There are several ways to handle this situation:

1. Incorporate the term \( \text{lt}(t, i, j, k, \ell) \) in the form of a tautology, as already described in (5) for the abstract disjunction;
2. Use disjunctive completion (see ??) to preserve the disjunction within the loop (which may ultimately lead to infinite disjunctions) or better allow only abstract predicates of the more restricted form \( r \lor \langle \text{lt}(t, m, n, p, q), r' \rangle \) (which definitively avoids the previous potential explosion);

3. Use semantically loop unrolling (as in [2, Sec. 6.5]) so that the loop:

\[
\text{while } B \text{ do } C \text{ od}
\]

is handled in the abstract semantics as if written in the form:

\[
\text{if } B \text{ then } C; \text{ while } B \text{ do } C \text{ od fi}
\]

which is equivalent in the concrete semantics. More generally, if several abstract terms of different kinds are considered (like \( \text{lt}(t, t, j, k, \ell) \) and \( s(t, m, n) \) in the forthcoming 17), a further semantic unrolling can be performed each time a term of a new kind does appear, while all terms of the same kind are merged by the widening.

Refined Generic Comparison Abstract Domains

- The generic comparison abstract domain \( D_{\text{lt}}(X) \) of 3.1 may be imprecise since it allows only for one term \( \langle \text{lt}(t, a, b, c, d), r \rangle \).
- First we could consider several arrays, with one such term per array.
- Second, we could consider the conjunction of such terms for a given array, which is more precise but may potentially lead to infinite conjunctions within loops (e.g. for which termination cannot be established).
- So we will consider this alternative within tests only, then applying the above abstract domain operators term by term.

For short we avoid to resort to semantical loop unrolling which is better adapted to automatization but would yield to lengthy handmade calculations in this section. This technique will be illustrated anyway in the forthcoming 17.

The same way we could the disjunctive completion of this domain, that is terms of the form \( \langle \text{lt}(t, a_{ij}, b_{ij}, c_{ij}, d_{ij}), r_{ij} \rangle \). This would introduce an exponential complexity factor, which we prefer to avoid. If necessary, we will use local trace partitioning [2, Sec. 6.6] instead.
Generic Comparison Static Program Analysis

Let us consider the following program (where \( a \leq b \)) which is similar to the inner loop of bubble sort [10]:

1: \[ \text{var } t : \text{array}[a,b] \text{ of int}; \]
2: \[ I := a; \]
3: while \((I < b)\) do
4: \[ \text{if } (t[I] > t[I + 1]) \text{ then} \]
5: \[ t[I] := t[I + 1] \]
6: \[ fi; \]
7: \[ I := I + 1 \]
8: \[ od \]

An Introduction to Abstract Interpretation, © P. Cousot, 26/3/03 — 3:26/58 — "|" | ▶ | ▼ | ▶ | Idx, Toc

Generic Choice of the Generic Relational Integer Abstract Domain

- We let \( P_p^1 \) be the value of the local predicate attached to the program point \( p = 1, \ldots, 8 \) at the \( i \)th iteration.
- Initially, \( P_0^0 = (a \leq b) \) while \( P_p^0 = \text{false} \) for \( p = 2, \ldots, 8 \).
- We choose the octagonal abstract domain [12, 13] as the generic relational integer abstract domain \( D_{rel}(X) \) parameterized by the set \( X \) of program variables \( I, J, \ldots \) and auxiliary variables \( t, j, \) etc.

Fixpoint Iterates

The fixpoint iterates are as follows:

\[
P_1^0 = (a \leq b) \quad \text{\{initialization to } P_0^0\} \]
\[
P_2^0 = (I = a \leq b) \quad \text{\{assignment } (I := a)\} \]
\[
P_3^0 = (I = a < b) \quad \text{\{loop condition } I < b\} \]
\[
P_4^0 = (lt(t, i, j, k, l), i = k = \ell = I = a < b \land j = I + 1) \quad \text{\{by (7) for test condition } (t[I] > t[I + 1])\}
\[
P_5^0 = (lt(t, m, n, p, q), \exists i, j, k, \ell: i = k = \ell = I = a < b \land j = I + 1 \land)
\]

\[
\text{\{by assignment } (10) \text{ which, by octagonal projection, simplifies into:}\]
\[
= (lt(t, m, n, p, q), m = I = a < b \land n = p = q = I + 1)
\]

\[ P_6^1 = (P_3^1 \land (lt(t, i, j, k, \ell), i = I = a < b \land j = k = \ell = I + 1)) \lor P_5^1 \]

\[ \text{\{by } (8) \text{ for test condition } (t[I] > t[I + 1]) \text{ and join } (9)\} \]
\[
= (lt(t, i, j, k, \ell), i = I = a < b \land j = k = \ell = I + 1) \lor \]
\[
(\forall m = I = a < b \land n = p = q = I + 1) \]

\[ \text{\{by def. } P_3^1 \text{ and } (2) \text{ as well as by def. of } P_5^1\} \]
\[
= (lt(t, a, b, c, d), (\exists i, j, k, \ell: a = i \land b = j \land c = k \land d = \ell \land i = I) \]

\[ \text{\{by def. } (4) \text{ of the abstract union } \lor \}
\]
\[
= (lt(t, a, b, c, d), (a = I = a < b \land b = c = d = I + 1) \lor (a = I = a \land \text{by octagonal projection}) \]
\[
= (lt(t, a, b, c, d), a = I = a < b \land b = c = d = I + 1) \quad \text{\{by octagonal disjunction\}}
\]

An Introduction to Abstract Interpretation, © P. Cousot, 26/3/03 — 3:27/58 — "|" | ▶ | ▼ | ▶ | Idx, Toc
\[ P_3^1 = \{ \text{lt}(t, a, b, c, d), a = I - 1 = a < b \land b = c = d = I \} \quad \text{by invertible assignment I := I + 1} \]

\[ P_3^2 = \{ \text{lt}(t, a, b, c, d), I = a + 1 = a + 1 \leq b \land b = c = d = I \} \quad \text{by octagonal simplification} \]

\[ P_3^3 = (P_3^1 \lor P_3^2) \land (I < b) \quad \text{by \text{lt}(t, a, b, c, d), I = a + 1 = a + 1 \leq b \land b = c = d = I} \]

\[ P_3^4 = (\text{lt}(t, i, j, k, \ell), i = j = k = \ell = I \leq a) \lor (I = i + 1 = a + 1 < b) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I \leq a + 1 \leq b} \]

\[ P_3^5 = (P_3^3 \lor P_3^4) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I < b} \]

\[ P_3^6 = (P_3^3 \lor P_3^4) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I < b} \]

\[ P_3^7 = (P_3^3 \land P_3^4) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I < b} \]

\[ P_3^8 = (P_3^3 \land P_3^4) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I < b} \]

\[ P_3^9 = (P_3^3 \land P_3^4) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I < b} \]

\[ P_3^{10} = (P_3^3 \land P_3^4) \quad \text{by \text{lt}(t, i, j, k, \ell), i = j = k = \ell = I < b} \]

Now the iterates have stabilized since:

\[ (P_3^3 \lor P_3^4) \land (I < b) \quad \text{since } P_3^3 = P_3^4 \text{ is stable} \]
\[
(I = a \leq b) \lor (\text{l$t(t,i,j,k,\ell)$, } i = a \leq j = k = \ell = I \leq b) \lor (I < b) \Downarrow \text{def. } P_2^3 \text{ and } P_2^3 \Downarrow
\]
\[
(I < b) \Downarrow \text{def. (5) of abstract disjunction with selection of } I \text{ as in (??)}
\]
\[
((\text{l$t(t,i,j,k,\ell)$, } (i = j = k = \ell = I = a \leq b) \lor (j = k = \ell = I = I < b) \lor (I < b) \Downarrow \text{by octagonal disjunction}) \Downarrow \text{by octagonal disjunction}) \Downarrow (\text{l$t(t,i,j,k,\ell)$, } i = a \leq j = k = \ell = I < b) \Downarrow (\text{l$t(t,i,j,k,\ell)$, } i = a \leq j = k = \ell = I < b) \Downarrow (\text{l$t(t,i,j,k,\ell)$, } i = a \leq j = k = \ell = I < b) \Downarrow \text{by abstract conjunction (2) } \Downarrow \text{by def. (1) of abstract implication}
\]
It remains to compute the loop exit invariant:
\[
(P_2^3 \lor P_3^3) \land (I \geq b)
\]

Then we define the generic sorting abstract domain:
\[
D_s(X) = \{ (s(t,a,b), r) \mid t \in X \land a, b \not\in X \land r \in D_{rel}(X \cup \{a, b\}) \} .
\]

The meaning \(\gamma((s(t,a,b), r))\) of an abstract predicate \((s(t,a,b), r)\) is, informally that the elements of \(t\) between indices \(a\) and \(b\) are sorted:

\[
\gamma((s(t,a,b), r)) = \exists a, b : t, \ell \leq a \leq b \leq \ell, r \land \forall i, j \in [a, b] : (i \leq j) \Rightarrow (t[i] \leq t[j]) \land r .
\]
**Generic Comparison and Sorting Abstract Domain**

The analysis of sorting algorithms involves the reduced product of the generic comparison abstract domain of 3.1 and sorting abstract domain of 14, that is triples of the form:

\[ (\text{lt}(t, a, b, c, d), s(t, e, f), r) \, . \]

**Reduction**

The reduction involves interactions between terms such as, e.g.:

\[ \text{lt}(t, a - 1, b - 1, b - 1) \land \text{lt}(t, a, b, b) \quad (15) \]
\[ \Rightarrow s(t, b - 1, b) \land \text{lt}(t, a, a - 1, b - 1, b) \]
\[ s(t, b + 1, c) \land \text{lt}(t, a, b + 1, b + 1, c) \land \text{lt}(t, a, b, b) \quad (16) \]
\[ \Rightarrow s(t, b, c) \land \text{lt}(t, a, b, b, c) \]
\[ \text{lt}(t, a + 1, a + 1, b) \land s(t, a + 1, b) \Rightarrow s(t, a, b) \quad (17) \]

The reduction [5] also involves the refinement of abstract predicate transformers (see a.o. [3, 11]) which would be performed automatically e.g. if the abstract predicate transformers are obtained by automatic simplification of the formula \( \alpha \circ F \circ \gamma \) (where \( F \) is the concrete semantics) by the simplifier of a theorem prover.

**Generic Comparison & Sorting Static Program Analysis**

Let us consider the bubble sort [10]:

```plaintext
1: var t: array [a, b] of int;
2: J := b;
3: while (a < J) do
4: I := a;
5: while (I < J) do
6: if (t[I] > t[I + 1]) then
7: t[I] := t[I + 1]
8: fi;
9: I := I + 1
10: od;
11: J := J - 1
12: od
```
**Fixpoint Approximation**

The fixpoint approximation is as follows ($P^i_p$ denotes the local assertion attached to program point $p$ at the $i$th iteration and $k$th loop unrolling, $P^i_p = P^i_p^{k,0}$ where $k = 0$ means that the decision to semantically unroll the loop is not yet taken):

\[ P^0_1 = (a \leq b) \quad \text{\textit{\{initialization\}}} \]

\[ P^i_1 = \text{false}, \quad i = 2, \ldots, 8 \]

\[ P^1_1 = P^0_1 \]

\[ = (a \leq b) \quad \text{\textit{\{def. $P^0_1$\}}} \]

\[ P^2_2 = (a \leq b = J) \quad \text{\textit{\{assignment $J := b$\}}} \]

\[ P^1_2 = (a < b = J) \quad \text{\textit{\{test $(a < J)$\}}} \]

\[ P^1_3 = (a < b = J) \quad \text{\textit{\{test $(a < J)$\}}} \]

\[ P^1_1 = s(t, J + 1, b) \land \text{lt}(t, a, J + 1, J + b) \land a < J = b - 1 \quad \text{\textit{\{by reduction (18)\}}} \]

\[ P^2_1 = s(t, J + 1, b) \land \text{lt}(t, a, J + 1, J + b) \land a < J = b - 1 \quad \text{\textit{\{by reduction (15)\}}} \]

\[ \text{as in 3.1 since the inner loop does (18) not modify $a$, $b$ or $I$ and the swap} \]

\[ t[1] := t[i + 1] \text{ does not interfere with} \]

\[ \text{lt}(t, a, J + 1, J + 1, J + 1) \text{ according to } a \leq I < I + 1 \leq J < J + 1 \text{ so } I, I + 1 \in [a, J + 1] \]

\[ \Rightarrow \text{lt}(t, a, J + 1, J + 1, J + 1) \land \text{lt}(t, a, J, J) \land a < J = b - 1 \quad \text{\{by elimination of $I$ is dead at program point 10\}} \]

\[ \Rightarrow s(t, J, b) \land \text{lt}(t, a, J, b) \land a < J = b - 1 \quad \text{\{by reduction (15)\}} \]

\[ P^{1,0}_{10} = \text{lt}(t, a, I, I, I) \land b < I = J \quad \text{\{as in 3.1 since the inner loop does not modify $a$, $b$ or $I$\}} \]

\[ \Rightarrow \text{lt}(t, a, J, J, b) \land a < b = J \quad \text{\{by elimination (octagonal projection) of program variable $I$ which is no longer live at program point 10\}} \]

\[ P^{1,0}_{11} = \text{lt}(t, a, J + 1, J + 1, b) \land a < b \land J = b - 1 \quad \text{\{postcondition for assignment $J := J - 1$\}} \]

\[ P^{1,1}_{11} = \text{lt}(t, a, J + 1, J + 1, b) \land a < J = b - 1 \quad \text{\{by semantical loop unrolling (since a new symbolic “lt” term has appeared, see 3.1), and test $(a < J)$\}} \]

\[ \ldots \]

\[ P^{1,1}_{10} = \text{lt}(t, a, J + 1, J + 1, J + 1) \land a < J = b - 1 \land \text{lt}(t, a, I, I, I) \land I = J \quad \text{\{by semantical loop unrolling (since a new symbolic “lt” term has appeared, see 3.1), and test $(a < J)$\}} \]
An Introduction to Abstract Interpretation, © P. Cousot, 25/3/03 — 3:45/58 — © | < | > |  ■ | ■ | ? | * | Idx, Toc

\[ P_{10}^{2,2} = s(t, J + 1, b) \land lt(t, a, J + 1, J + 1, b) \land a < J \leq b - 2 \land lt(t, a, I, I, I) \land I = J \] (by 3.1 and non interference, see (18))

Now \( P_{11}^{2,2} \land a < J \Rightarrow P_{12}^{3,2} \) so that the loop iterates stabilize to a post-fixpoint. On loop exit, we must collect all cases following from semantic unrolling:

\[ P_{12}^{3,2} = (P_{12}^3 \land a \geq J) \] (no entry in the loop)

\[ \lor (P_{12}^{1,0} \land a \geq J) \] (loop exit after one iteration)

\[ \lor (P_{11}^{1,0} \land a \geq J) \] (loop exit after two iterations)

\[ \lor (P_{11}^{2,2} \land a \geq J) \] (loop exit after three iterations or more)
Conclusion

- Observe that **generic predicate abstraction** is defined for a programming language as opposed to **ground predicate abstraction** which is specific to a program, a usual distinction between abstract interpretation based static program analysis (a generic abstraction for a set of programs) and abstract model checking (an abstract model for a given program).
- Notice that the so-called **polymorphic predicate abstraction** of [1] is an instance of symbolic relational separate procedural analysis [6, Sec. 7] for **ground** predicate abstraction.
- The generalization to generic predicate abstraction is immediate since it only depends on the way concrete predicate transformers are defined (see [6, Sec. 7]).

Bibliography


THE END