# Ultimate Taipan

ultimate.informatik.uni-freiburg.de

github.com/ultimate-pa/ultimate

**Daniel Dietsch, Marius Greitschus, Matthias Heizmann, Jochen Hoenicke, Alexander Nutz, Andreas Podelski, Christian Schilling, Tanja Schindler**

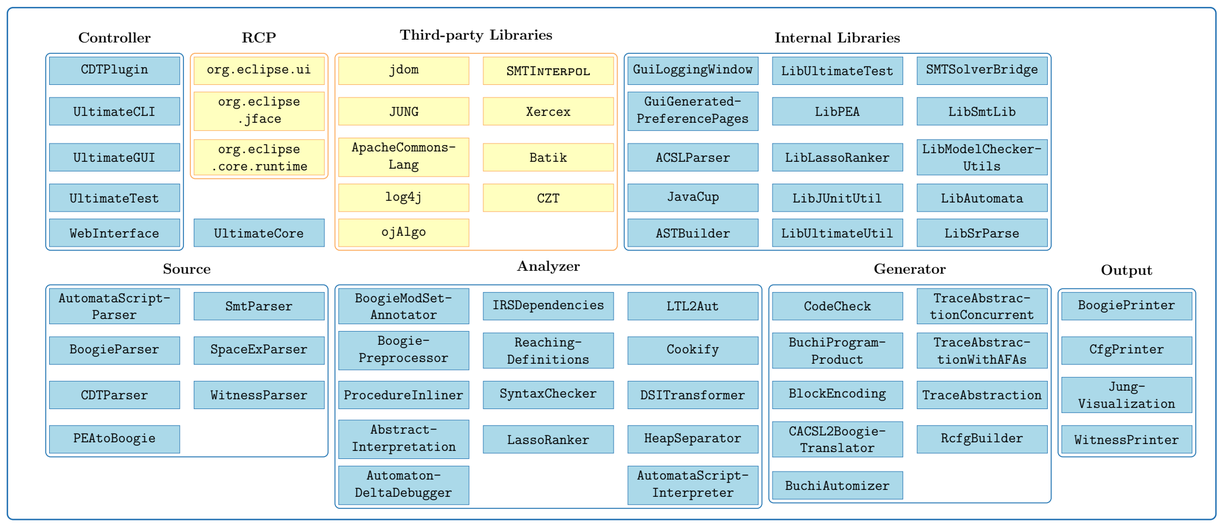
## Definitions

- $\mathcal{A}_\mathcal{P}$: Program automaton
- $\mathcal{A}_D$: Data automaton
- $\mathcal{H}$: Set of Hoare triples
- $htc^\#$: Hoare triples are checked with the abstract post transformer.
- $htc^{TA}$: Hoare triples are checked with SMT solver.

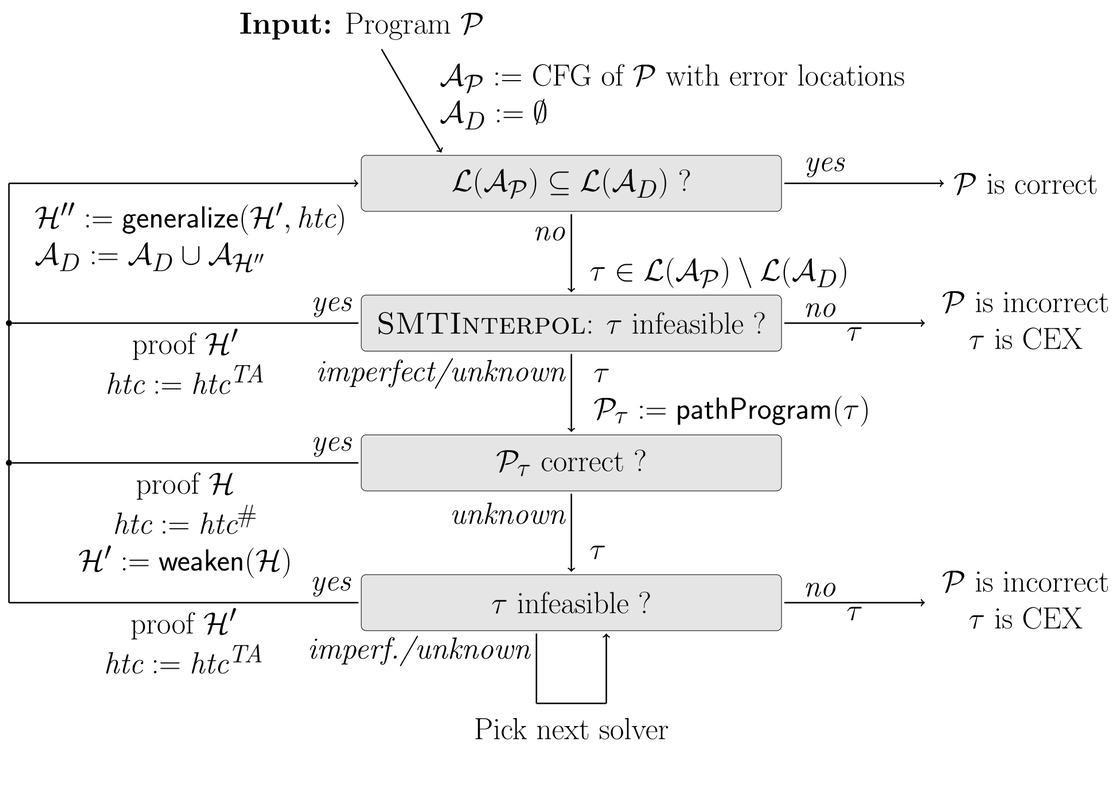
### Perfect Interpolants

A sequence of interpolants is perfect if the state assertions are not only inductive with respect to the trace, but also to the path program induced by the trace.

## Ultimate Program Analysis Framework




## Ultimate Taipan Workflow




## generalize$(\mathcal{H}, htc)$

**for** $\mathtt{s} \in \mathit{Statements}$ **do**
  **for** $\varphi, \varphi' \in \mathit{Predicates}(\mathcal{H})$ **do**
    **if** $htc(\{\varphi\}\,\mathtt{s}\,\{\varphi'\}) = \top$ **then** $\mathcal{H} := \mathcal{H} \cup \{\{\varphi\}\,\mathtt{s}\,\{\varphi'\}\}$
  **end**
**end**

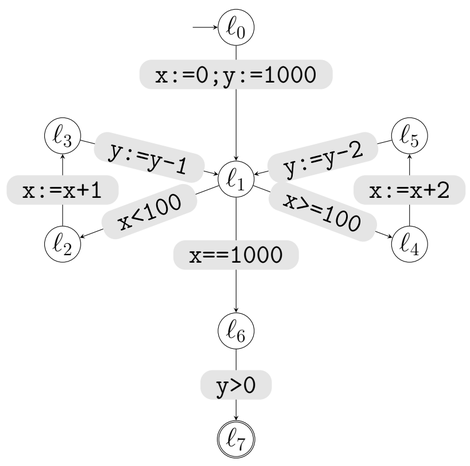
## weaken$(\mathcal{H})$

$K := \emptyset$, $\mathcal{H}' := \emptyset$, $\hat{\mathcal{H}} :=$ reverse sequence of Hoare triples in $\mathcal{H}$, $\varphi_p := \mathit{false}$
**for** $\{\varphi\}\mathtt{s}\{\varphi'\} \in \hat{\mathcal{H}}$ **do**
  $W_\mathtt{s} :=$ All variables only written in $\mathtt{s}$
  $R_\mathtt{s} :=$ All variables read in $\mathtt{s}$
  $K := K \setminus W_\mathtt{s}$
  $K_\mathtt{s} := R_\mathtt{s} \cup K$
  $\hat{\varphi} := \varphi$ without conjuncts that contain only variables not in $K_\mathtt{s}$
  $K := K \cup \mathit{Variables}(\hat{\varphi})$
  $\mathcal{H}' := \mathcal{H}' \cup \{\{\hat{\varphi}\}\,\mathtt{s}\,\{\varphi_p\}\}$, $\varphi_p := \hat{\varphi}$
**end**

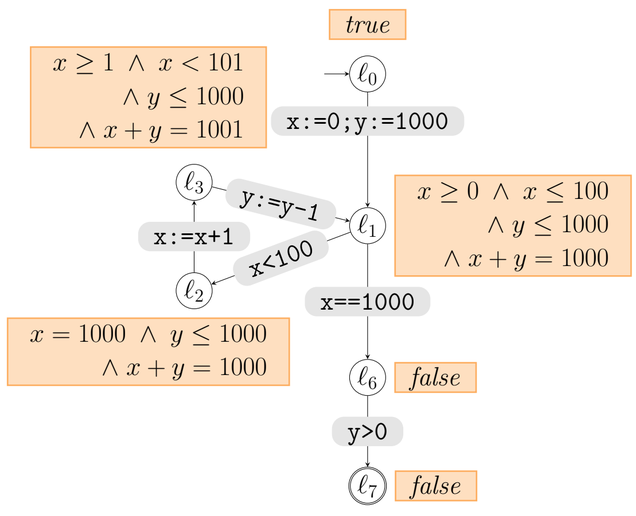
## Example

```
1  int x:=0, y:=1000;
2  while(*){
3    if (x < 100) {
4      x:=x+1;
5      y:=y−1;
6    } else {
7      x:=x+2;
8      y:=y−2;
9    }
10 }
11 if (x==1000) {
12   assert y<=0;
13 }
```

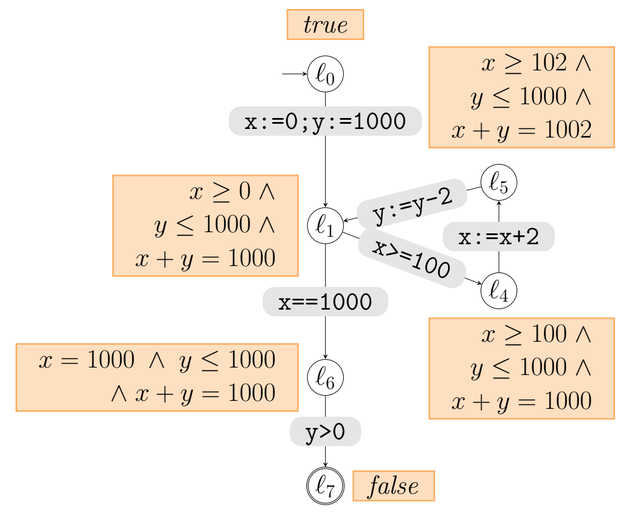
Example program.




Corresponding CFG.




Path program $\mathcal{P}_{\tau_1}$.




Path program $\mathcal{P}_{\tau_2}$.

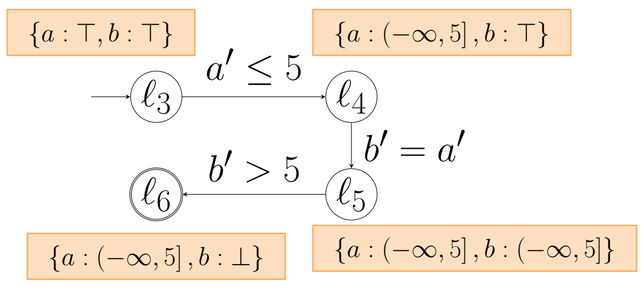
$\tau_1 :=$ `x:=0;y:=1000` `x<100` `x:=x+1` `y:=y+1` `x==1000` `y>0`

$\tau_2 :=$ `x:=0;y:=1000` `x>=100` `x:=x+2` `y:=y+2` `x==1000` `y>0`

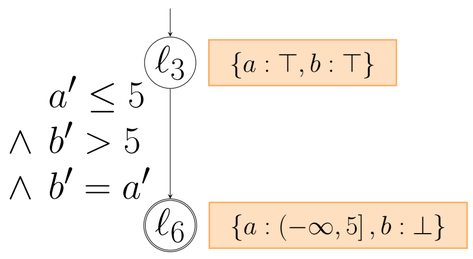
## Dynamic Block Encoding

```
1 procedure foo() {
2   var a, b : int;
3   assume a <= 5;
4   assume b == a;
5   assert b <= 5;
6 }
```

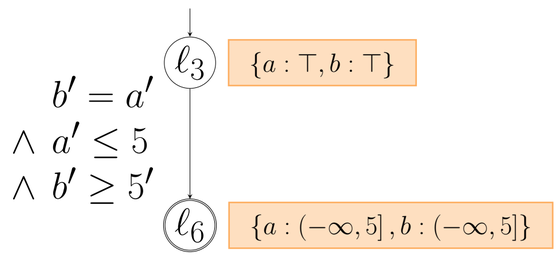
Example program.




No block encoding.




Large block encoding A.




Large block encoding B.

- Expressibility $ex$: Predicate that decides for a subformula whether it can be represented precisely by the abstract domain.
- $post^\#$: Abstract transformer which computes an abstract post state $\sigma'$ from a given abstract pre-state $\sigma$ and a transition formula $\varphi$.
- $\bigsqcap$ and $\bigsqcup$ are the meet and join operators.
- We convert transition formulas to DNF, i.e.,
  $\varphi = \varphi_0 \vee \varphi_1 \vee \ldots \vee \varphi_n$ and $\varphi_i = \varphi_i^0 \wedge \varphi_i^1 \wedge \ldots \wedge \varphi_i^m$.

**Algorithm**:

1. Partition conjuncts by expressibility predicate $ex$.
2. Compute $post^\#$ for all expressible conjuncts: $\sigma_i'' = \bigsqcap_{ex(\varphi_i^k)} post^\#(\sigma, \varphi_i^k)$.
3. Compute $post^\#$ repeatedly for all non-expressible conjuncts until a fixpoint is found starting with $\sigma_i'' : \sigma_i' = fp\left(\bigsqcap_{\neg ex(\varphi_i^k)} post^\#(\sigma_i'', \varphi_i^k)\right)$.
4. Compute 1 – 3 for each disjunct in $\varphi$ and combine resulting $\sigma_i'$ as usual: $post^\#(\sigma, \varphi) = \bigsqcup_{i=0}^n \sigma_i'$.