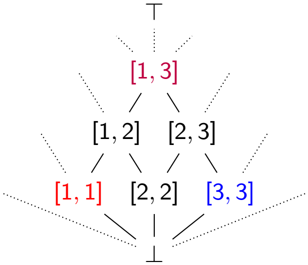
DPLL-Style Program Analysis

Leopold Haller

POPL Student Blitz Session

Imprecision in Abstract Interpretation

- Abstract interpretation sound but not complete.
- Incompleteness manifests in imprecision during the analysis.



Example: Domain of Intervals

Imprecisions in the Domain

Imprecision in join

```
x:=*;
if(x > 5)
  y := -1;
else
  y := 1;

assert(y != 0);
```

$\longrightarrow y \in [-1, -1], x \in [6, \infty]$

$\longrightarrow y \in [1, 1], x \in [-\infty, 5]$

$\longrightarrow y \in [-1, 1]$

The disjunction $y = 1 \vee y = -1$ cannot be expressed as an interval.

Imprecisions in the Domain

Imprecision in join

```
x:=*;
if(x > 5)
  y := -1;
else
  y := 1;

assert(y != 0);
```

$\longrightarrow y \in [-1, -1], x \in [6, \infty]$ (after `y := -1;`)

$\longrightarrow y \in [1, 1], x \in [-\infty, 5]$ (after `y := 1;`)

$\longrightarrow y \in [-1, 1]$ (at `assert(y != 0);`)

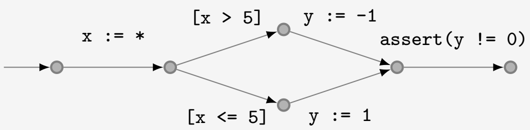
The disjunction $y = 1 \vee y = -1$ cannot be expressed as an interval.

How can we introduce disjunctions just where we need them?

Trace Partitioning

- Consider separately different sets of traces through a program
- Think: Case splits in a proof.

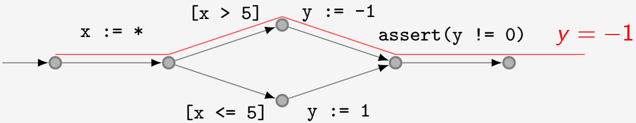
Control-flow based trace partitioning



Trace Partitioning

- Consider separately different sets of traces through a program
- Think: Case splits in a proof.

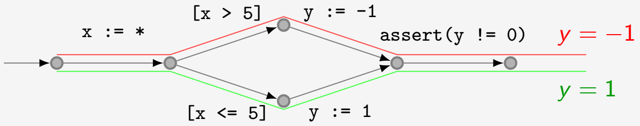
Control-flow based trace partitioning



Trace Partitioning

- Consider separately different sets of traces through a program
- Think: Case splits in a proof.

Control-flow based trace partitioning



The main question is:

The main question is:
How can we find a good partitioning?

The main question is:
How can we find a good partitioning?

just precise enough
abstract enough to be efficient

Clipped fixpoints

Standard analysis

$$\hat{FP} \equiv \mu X.I \sqcup \hat{F}(X)$$

This may be too imprecise for the reasons mentioned earlier.

Clipped fixpoints

Standard analysis

$$\hat{FP} \equiv \mu X . I \sqcup \hat{F}(X)$$

This may be too imprecise for the reasons mentioned earlier.

Clippings

Find a set $a_1, \ldots, a_k$ of abstract elements and compute for each $1 \leq i \leq k$

$$\hat{FP}_i \equiv \mu X . I \sqcup (\hat{F}(X) \sqcap a_i)$$

such that each program behaviour is represented in some $\hat{FP}_i$.
Any checks can be performed on the FP_i for increased precision.

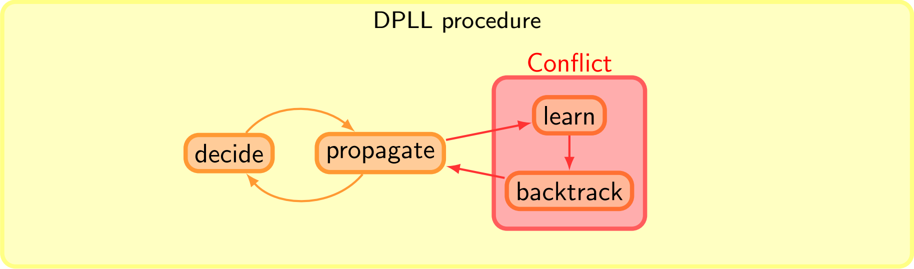
Clippings are equivalent to a certain class of trace partionings

Reframed question:

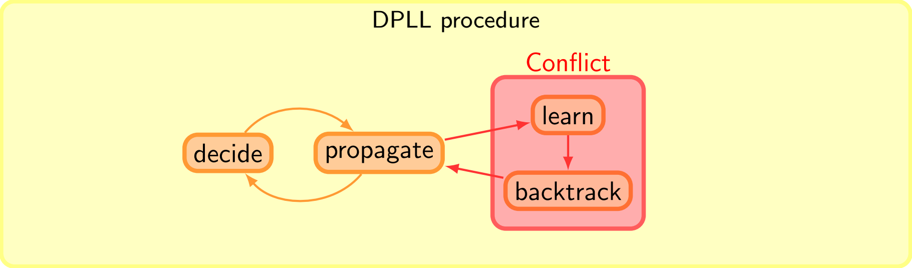
Reframed question:
How do we find these elements $a_1, \ldots, a_k$?

Let's look at an architecture that's good at dealing with disjunction

DPLL architecture

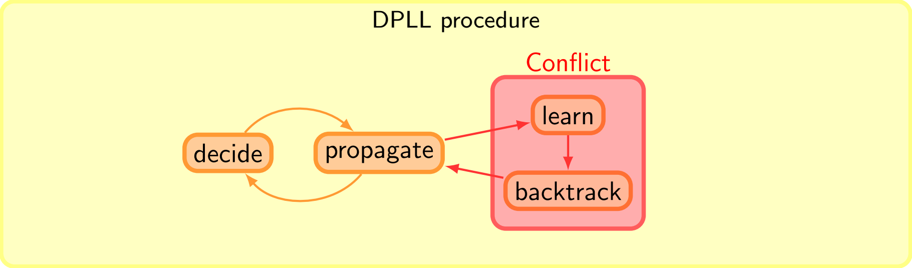


DPLL architecture



- Main phases of the DPLL procedure:

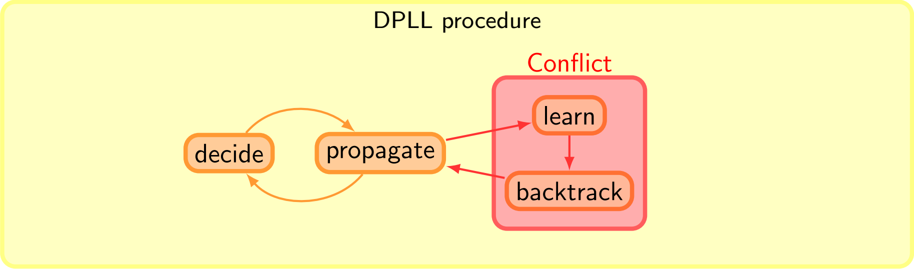
DPLL architecture



- Main phases of the DPLL procedure:

  Decision Assume a value for an undetermined variable

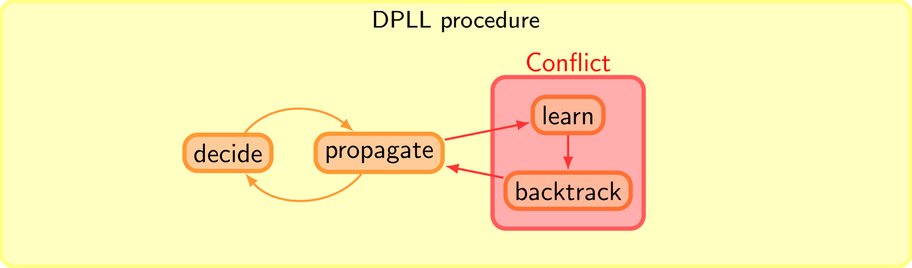
DPLL architecture



- Main phases of the DPLL procedure:

Decision	Assume a value for an undetermined variable
Propagation	Deduce implied variable values

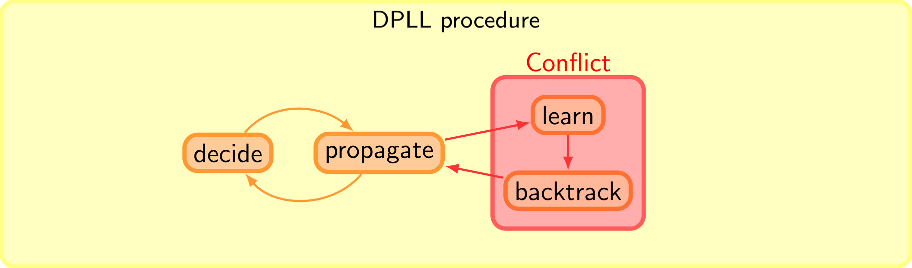
DPLL architecture



- Main phases of the DPLL procedure:

Decision	Assume a value for an undetermined variable
Propagation	Deduce implied variable values
Learning	Learn reason for conflict and backtrack

DPLL architecture

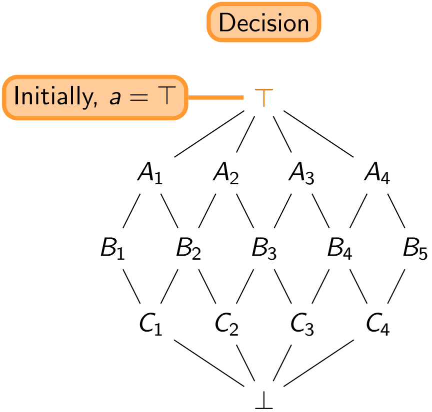


- Main phases of the DPLL procedure:

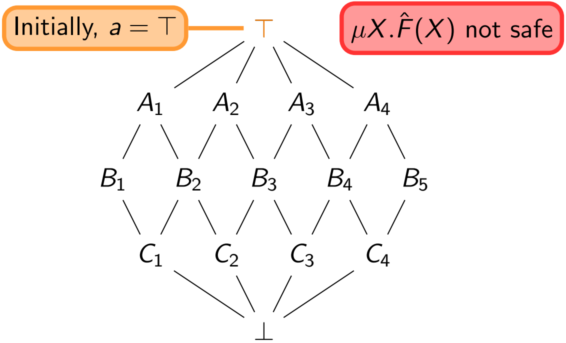
Decision	Assume a value for an undetermined variable
Propagation	Deduce implied variable values
Learning	Learn reason for conflict and backtrack

Use same architecture for program analysis. Current variable assignment corresponds to clipping.

SAT-Style Program Analysis



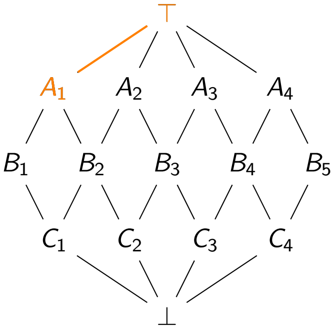
SAT-Style Program Analysis



SAT-Style Program Analysis

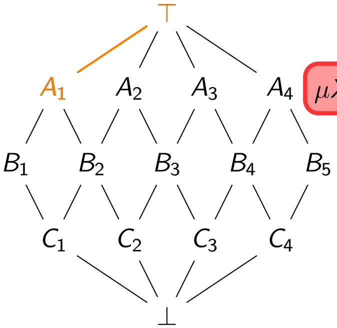
Decision

Decision: refine a



SAT-Style Program Analysis

Decision: refine a

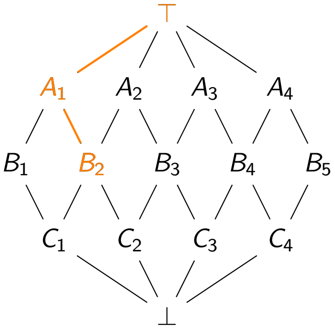


$\mu X.(\hat{F}(X) \sqcap A_1)$ not safe

SAT-Style Program Analysis

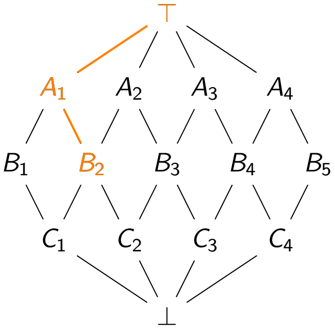
Decision

Decision: refine a



SAT-Style Program Analysis

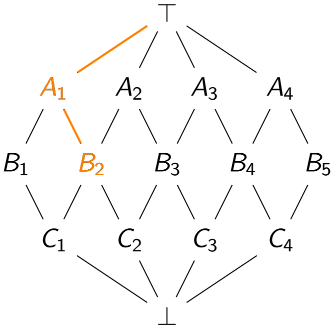
Decision: refine a



$\mu X.(\hat{F}(X) \sqcap B_2)$ safe

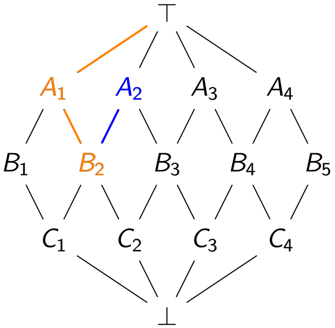
SAT-Style Program Analysis

Generalization



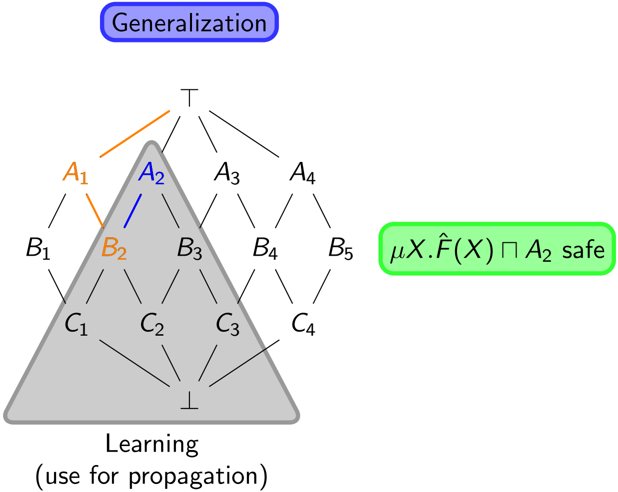
SAT-Style Program Analysis

Generalization



$\mu X.\hat{F}(X) \sqcap A_2$ safe

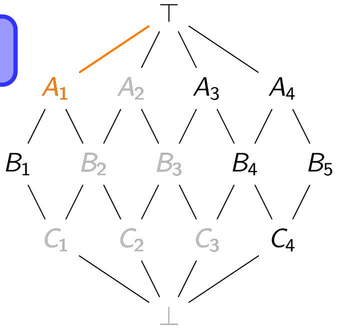
SAT-Style Program Analysis



SAT-Style Program Analysis

Generalization

Backtrack
and continue



Summary & Application

- Refine domain in a property dependent way by using a DPLL style analysis.
- Application to verification of industrial floating-point programs using value-based partitionings

Summary & Application

- Refine domain in a property dependent way by using a DPLL style analysis.
- Application to verification of industrial floating-point programs using value-based partitionings

Thanks for your attention.