The Model Evolution Calculus

Jorge Baier

Based on slides by Peter Baumgartner

Background

- An effective method for SAT was pioneered by Davis, Putman, Logemann, and Loveland (DPLL).

- The best modern SAT solvers (MiniSat, zChaff, Berkmin,…) are based on DPLL.

- FDPLL lifts DPLL to first-order but still does not emulate DPLL faithfully.

What is the ME

The **M**odel **E**volution Calculus (ME)

≈

First-Order DPLL

+ DPLL's simplification rules

+ Universal variables

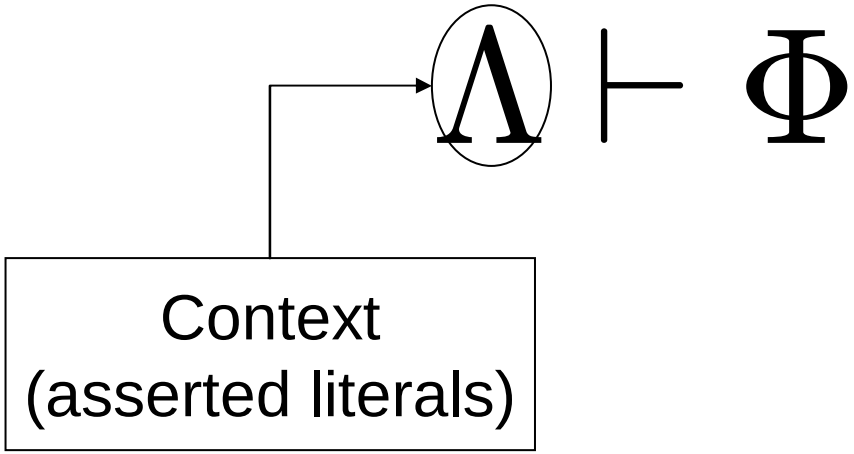
The calculus is a direct lifting of the **whole DPLL** to the first-order level.

Overview

- The DPLL method as a sequent-style calculus
- The Model Evolution calculus as a lifting of the DPLL calculus
- Properties of the ME calculus
- Implementation

The DPLL Calculus

$$\Lambda = \{literals\} \qquad \Phi = \{clauses\}$$



DPLL Calculus (cont.)

- If we want to prove that Φ is (un)satisfiable, we start with the sequent $\emptyset \vdash \Phi$

- By the successive application of inference rules we try to obtain either that

$$\Lambda \vdash \emptyset \quad (\Phi \text{ is SAT and } \Lambda \text{ is a model})$$

or

$$\Lambda \vdash \square \quad (\Phi \text{ is UNSAT})$$

The DPLL Calculus

$\Lambda = \{literals\}$ $\Phi = \{clauses\}$

L literal $\square$ empty clause C clause

The Split rule:

$$\frac{\Lambda \vdash \Phi, L \vee C}{\Lambda, L \vdash \Phi, L \vee C \qquad \Lambda, \overline{L} \vdash \Phi, L \vee C}$$

if

$$\begin{cases} C \neq \square, \\ L \notin \Lambda, \overline{L} \notin \Lambda \end{cases}$$

The DPLL Calculus

$\Lambda = \{literals\}$ $\Phi = \{clauses\}$

L literal $\square$ empty clause C clause

$$(\text{subsume})\ \frac{\Lambda \vdash \Phi, L \vee C}{\Lambda \vdash \Phi}\ \text{if}\ L \in \Lambda$$

$$(\text{resolve})\ \frac{\Lambda \vdash \Phi, L \vee C}{\Lambda \vdash \Phi, C}\ \text{if}\ \overline{L} \in \Lambda$$

$$(\text{close})\ \frac{\Lambda \vdash \Phi, L_1 \vee \ldots \vee L_n}{\Lambda \vdash \square}\ \text{if}\ \overline{L_1}, \ldots, \overline{L_n} \in \Lambda$$

The DPLL Calculus (cont.)

$\Lambda = \{literals\}$ $\Phi = \{clauses\}$

L literal $\square$ empty clause C clause

The Assert rule:

$$\frac{\Lambda \vdash \Phi, L}{\Lambda, L \vdash \Phi, L} \text{ if } \begin{cases} L \notin \Lambda, \\ \overline{L} \notin \Lambda \end{cases}$$

The DPLL Calculus: Key Insight

$$\Lambda \vdash \Phi$$

Λ can be seen as a finite representation of a Herbrand interpretation:

$$I_\Lambda = \{ L \in \Lambda \mid L \text{ is positive} \}$$

If I_Λ does not satisfy Φ,
"repair" it by adding literals to Λ

Some Notation

$I_\Lambda \Vdash \varphi$ iff $I_\Lambda \models \varphi$ and there is no $\Lambda' \supseteq \Lambda$ s.t. $I_{\Lambda'} \models \neg\varphi$

permanently satisfies

Examples:

$I_{\{p\}} \models \neg p \vee \neg q$ but not $I_{\{p\}} \Vdash \neg p \vee \neg q$

$I_{\{\neg p\}} \models \neg p \vee \neg q$ and $I_{\{\neg p\}} \Vdash \neg p \vee \neg q$

An alternative Split Rule

$$(\text{split})\ \frac{\Lambda \vdash L \vee C, \Phi}{\Lambda, L \vdash L \vee C, \Phi \qquad \Lambda, \overline{L} \vdash L \vee C, \Phi}\ \text{if } (*)$$

$$(*) \quad = \quad \begin{cases} 1)\ \text{not } I_\Lambda \models L \vee C \\ 2)\ L \text{ not contradictory with } \Lambda \\ 3)\ \overline{L} \text{ not contradictory with } \Lambda \end{cases}$$

The DPLL Calculus Revisited: A Model Evolution View

$$(\text{resolve})\ \frac{\Lambda \vdash \Phi, L \vee C}{\Lambda \vdash \Phi, C}\ \text{if}\ I_\Lambda \| \models \overline{L}$$

Lifting DPLL to First Order Logic

Main questions:

- How to represent Herbrand models. (we already know that)
- How to check $\models$
- How to check $\models\!\!\models$
- How to repair an interpretation

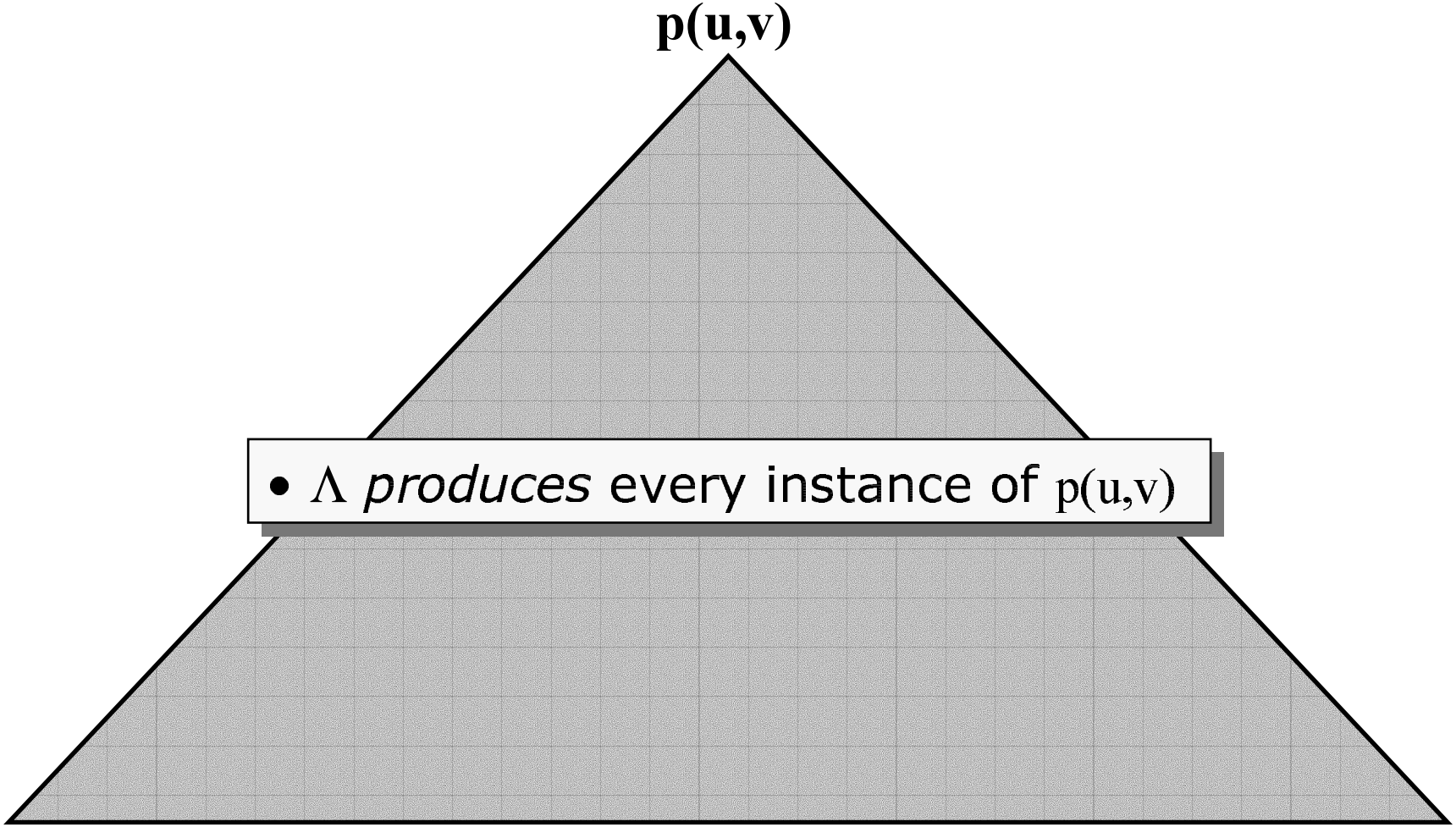
First-order Contexts

Sets Λ of parametric literals $L(u,v,..)$ and universal literals $L(x,y,\ldots)$

- parameters $(u,v, \ldots)$ and variables $(x,y,\ldots)$ both stand for ground terms
- (roughly) a parametric literal L in Λ denotes all of its ground instances, unless $\neg L' \in \Lambda$ for some instance L' of L
- a universal literal denotes all of its ground instances, unconditionally

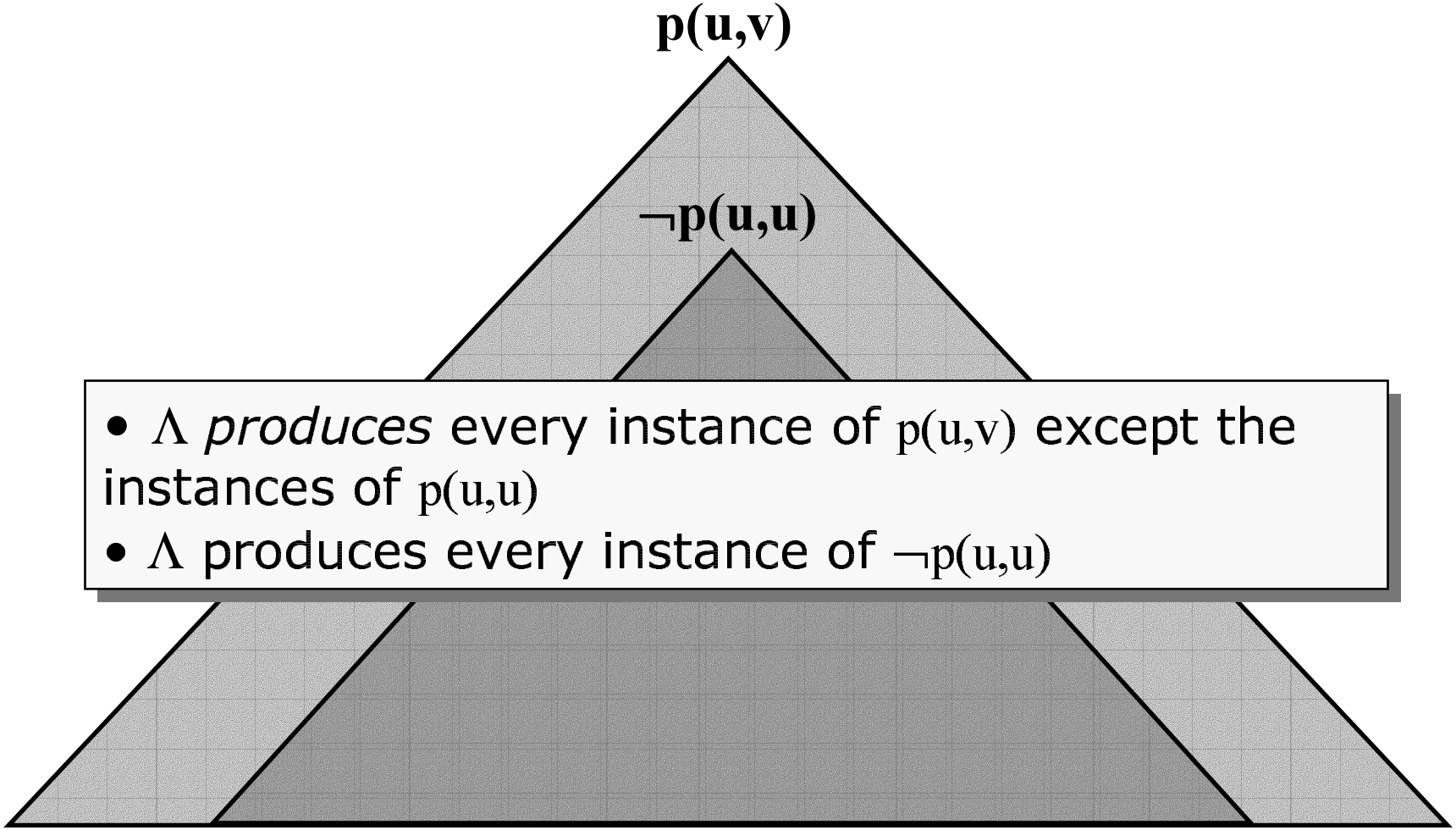
First-order Contexts: Examples

$\Lambda = \{ p(u,v) \}$



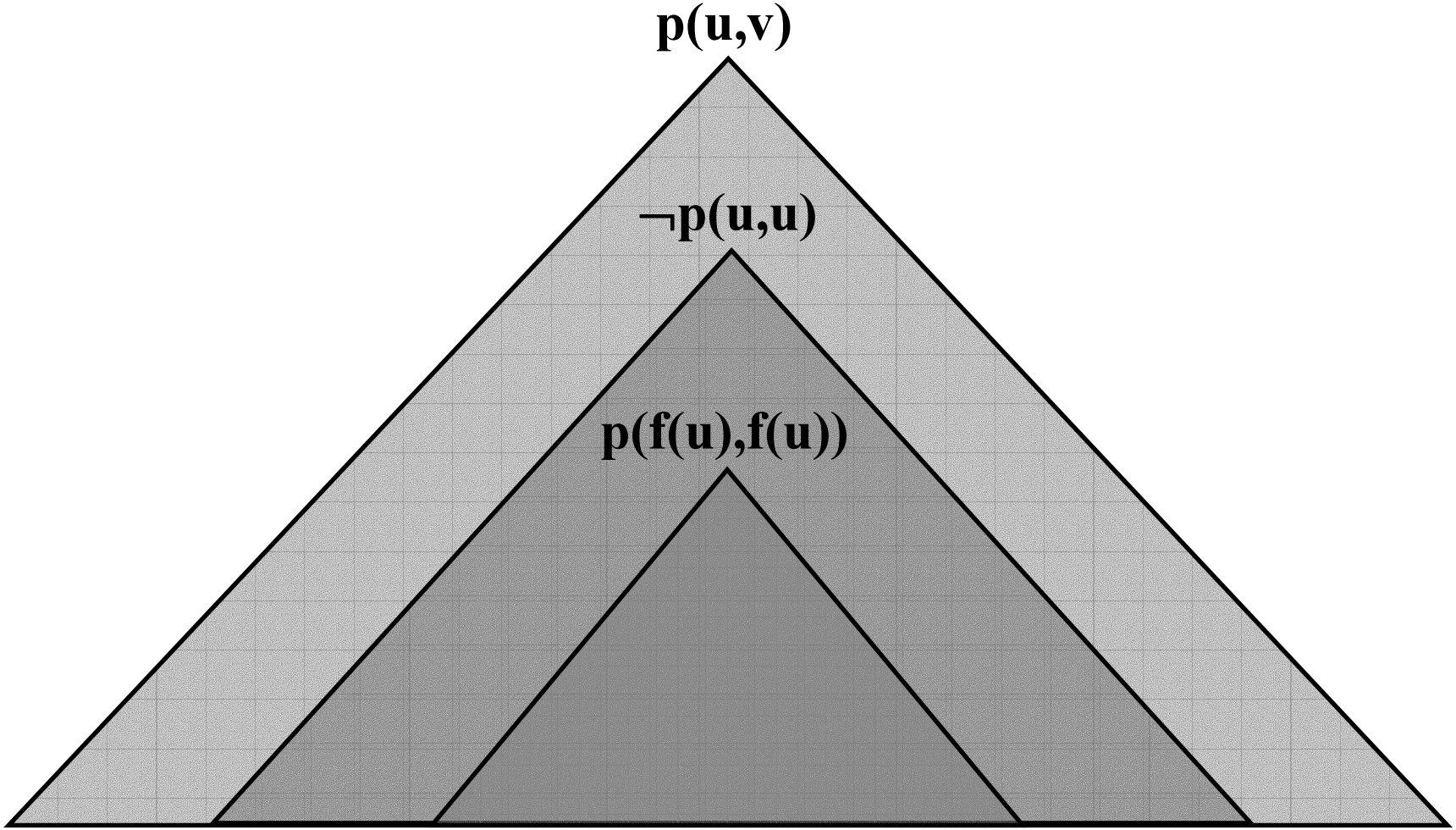
First-order Contexts: Examples

$\Lambda = \{p(u,v), \neg p(u,u)\}$



First-order Contexts: Examples

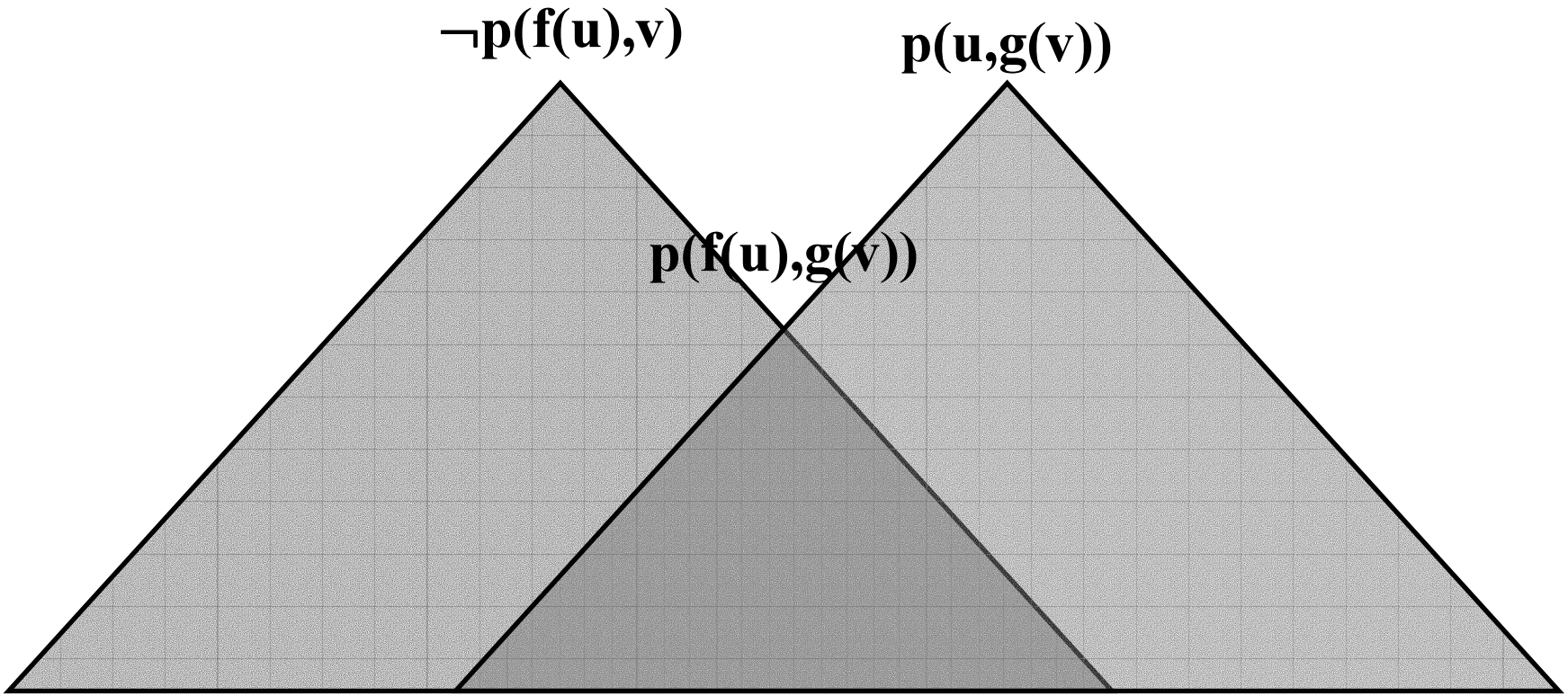
$\Lambda = \{p(u,v), \neg p(u,u), p(f(u),f(u))\}$



First-order Contexts: Examples

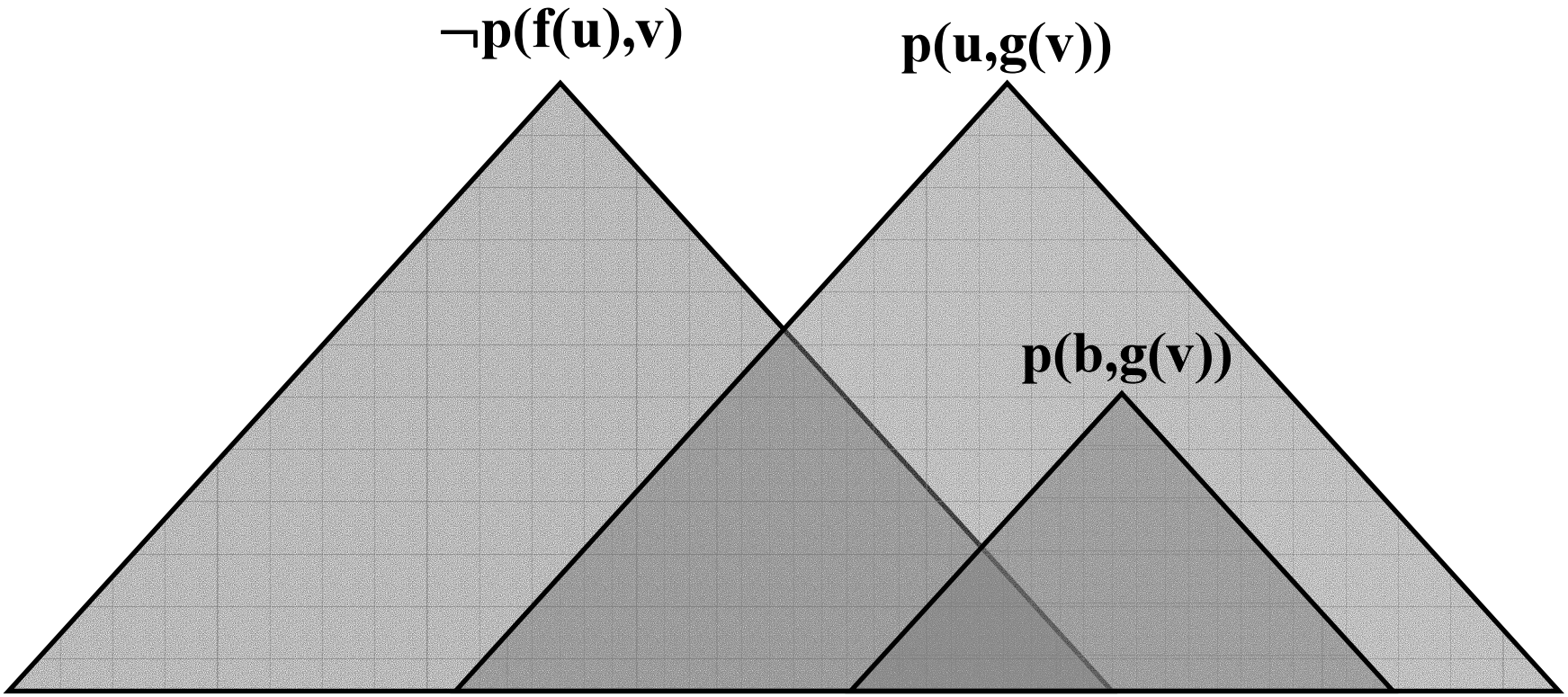
$\Lambda = \{\neg p(f(u),v),\ p(u,g(v))\}$

OK



First-order Contexts: Examples

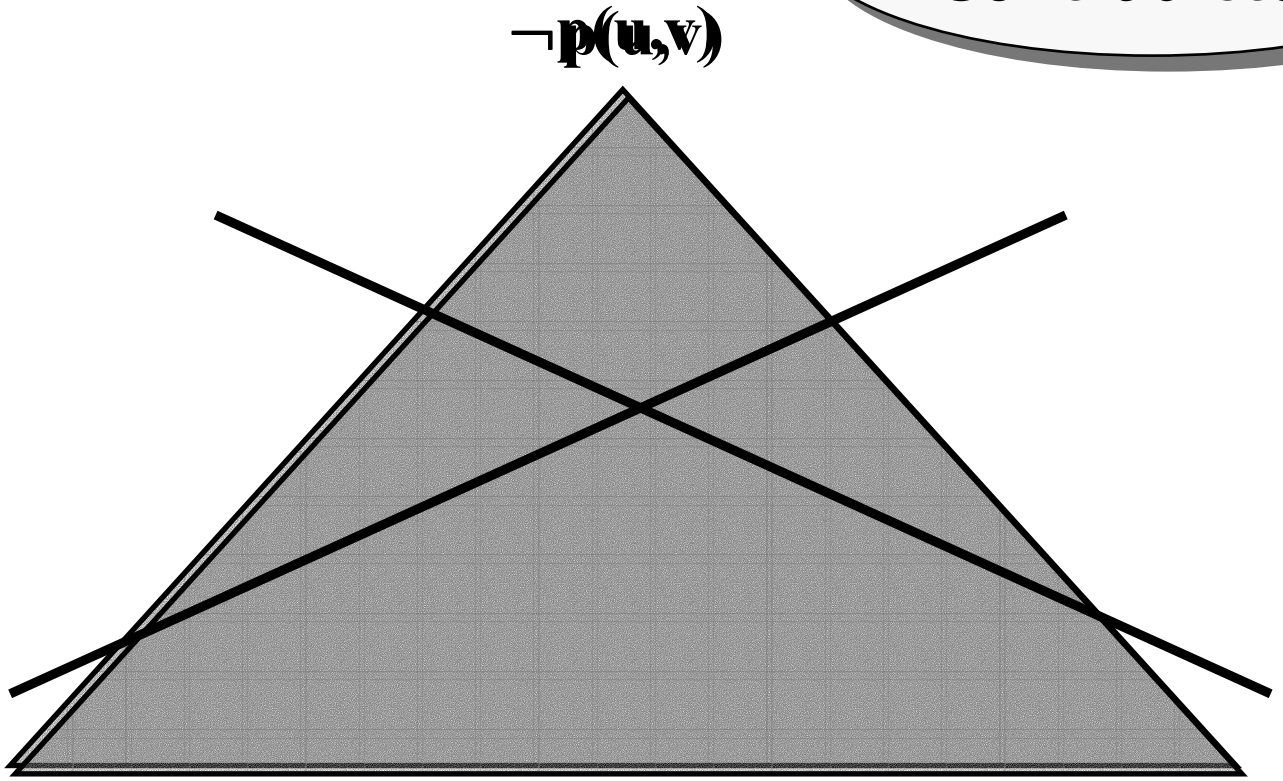
$\Lambda = \{\neg p(f(u),v),\ p(u,g(v)),\ p(b,g(v))\ \}$ OK



First-order Contexts: Examples

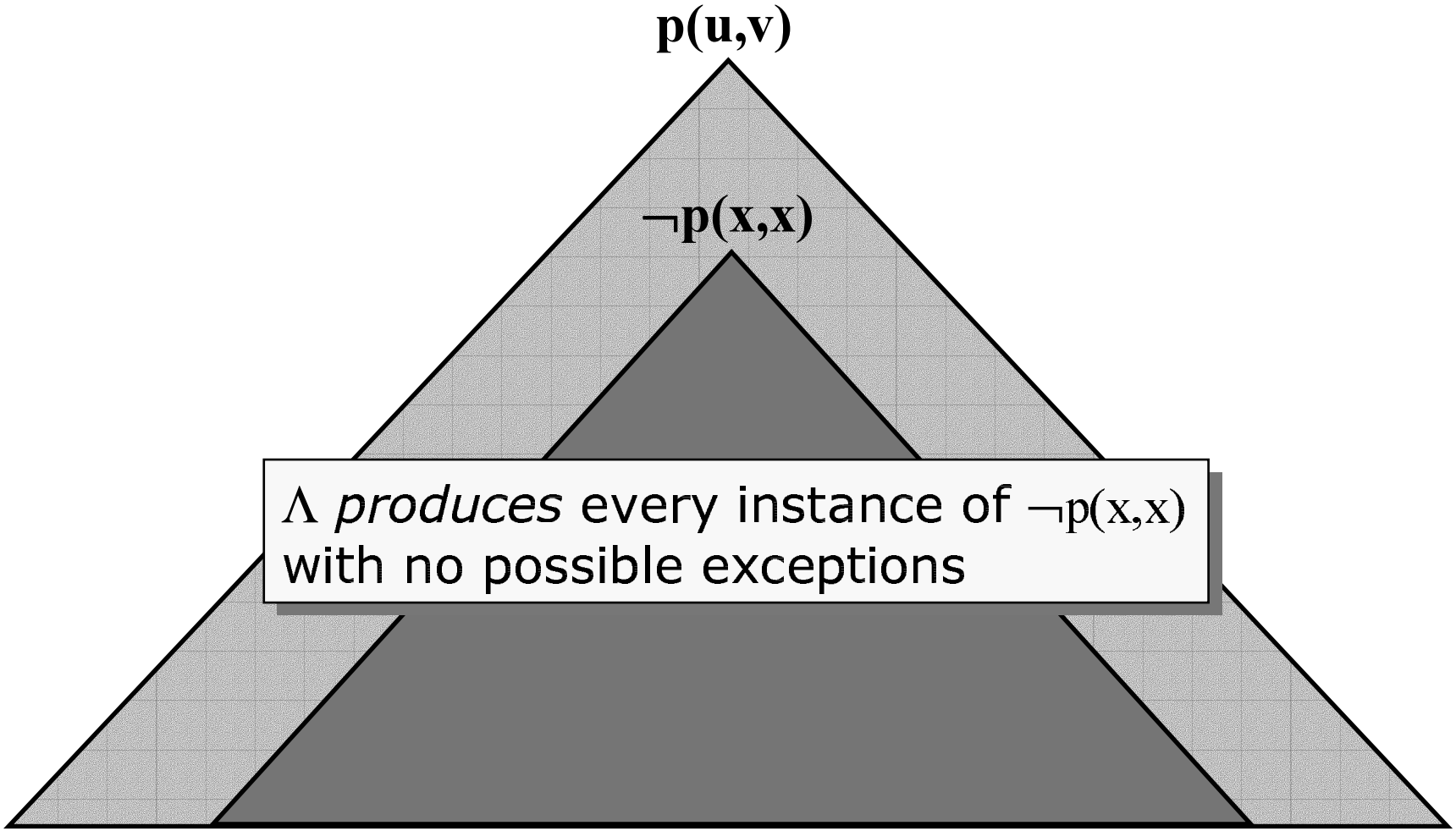
$\Lambda = \{\neg p(u,v), p(u,v)\}$

Not OK!
Contradictory



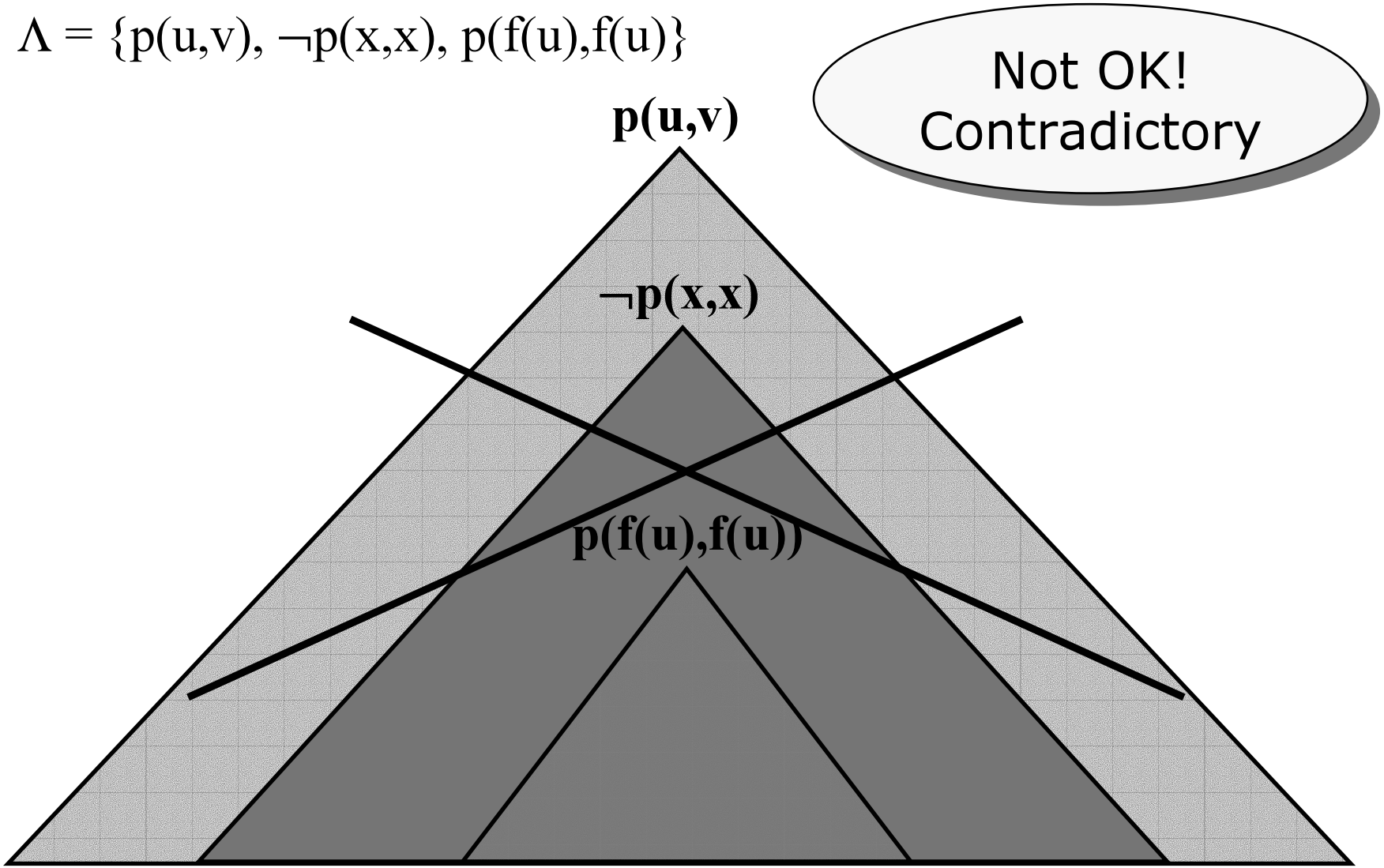
First-order Contexts: Examples

$\Lambda = \{p(u,v), \neg p(x,x)\}$



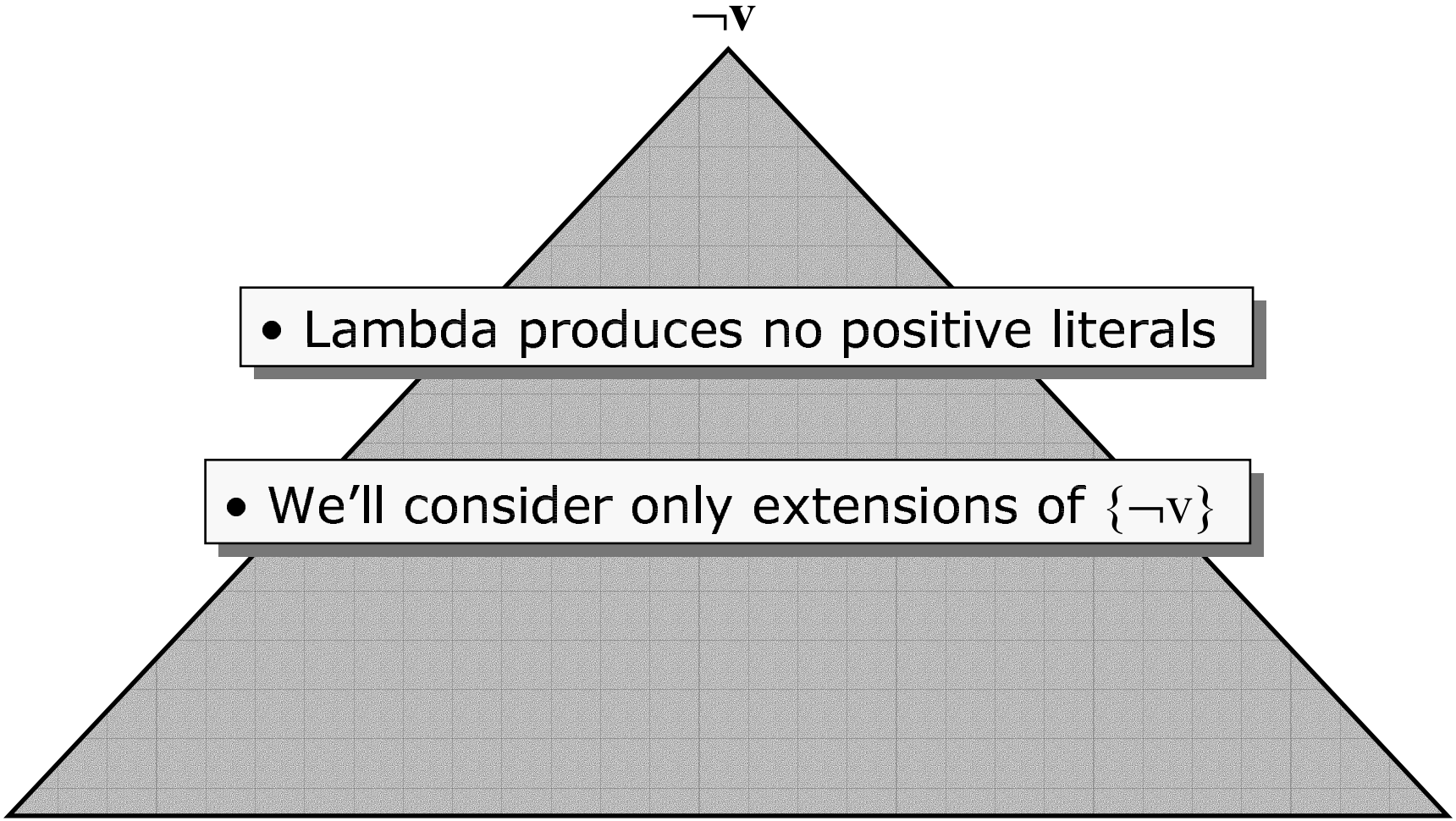
First-order Contexts: Examples

$\Lambda = \{p(u,v), \neg p(x,x), p(f(u),f(u)\}$



Initial Context

$\Lambda = \{\neg v\}$



Contexts and Interpretations

Let Λ be a non-contradictory context with parametric literals and universal literals

Λ denotes a Herbrand interpretation:

$$I_\Lambda = \left\{ L \;\middle|\; \begin{array}{l} L \text{ is ground and positive,} \\ L \text{ is produced by } \Lambda \end{array} \right\}$$

Checking $\models$

Let Λ be a non-contradictory context

Let $L_1 \vee \ldots \vee L_n$ be a (parameter-free) clause

If not $I_\Lambda \models L_1 \vee \ldots \vee L_n$ then

there are fresh variants $K_1, \ldots, K_n$ of literals in Λ

and a substitution σ

such that

σ is a simultaneous mgu of $\{K_1, \overline{L_1}\}, \ldots, \{K_n, \overline{L_n}\}$

If K_i produces $\overline{K_i\sigma}$, then $L_i\sigma$ is part of the **remainder** of C

If $P_{\Lambda}(\overline{K_i\sigma}) \not\simeq$ *true*, then $L_i\sigma$ is part of the *productive*

- σ is called a *context unifier* (of the clause against Λ)

Context Unifiers: an example

Let $\Lambda = \{\neg v, p(v_1, u_1), \neg p(x_1, g(x_1)), q(v_2, g(v_2))\}$

$$C = r(x) \vee \neg p(x, y)$$

The substitution $\sigma = \{v \to r(v_1), x \to v_1, y \to u_1\}$
is a context unifier of C against Λ
The remainder is $r(v_1) \vee \neg p(v_1, u_1)$

A context unifier is **admissible for Split** if all literals in the remainder are parameter-free or variable-free. Moreover, they must not share variables

Checking $\|\!=$ (not sure about this!)

Let Λ be a non-contradictory context

Let $L_1 \vee \ldots \vee L_n$ be a (parameter-free) clause

$I_\Lambda \|\!= \neg(L_1 \vee \ldots \vee L_n)$

iff there are fresh variants $K_1, \ldots, K_n$ of literals in Λ

and a substitution σ

such that

1. σ is a simultaneous mgu of $\{K_1, \overline{L_1}\}, \ldots, \{K_n, \overline{L_n}\}$
2. for each i, $Pars(K_i)\sigma \subseteq Pars$

The remainder is empty!

The Model Evolution Calculus: Semantical View

$$\text{(assert)}\ \frac{\Lambda \vdash \Phi, L}{\Lambda, L \vdash \Phi, L}\ \text{if (*)}$$

(*) no $K \in \Lambda$ is s.t. $K \geq L$ and L is not contradictory with Λ

$$\text{(assert)}\ \frac{\Lambda \vdash \Phi, C \vee L}{\Lambda, L \vdash \Phi, C \vee L}\ \text{if (**)}$$

(**) There is a context unifier σ of C with an empty remainder and
$L\sigma$ is not contradictory with Λ and
no $K \in \Lambda$ is s.t. $K \geq L\sigma$

The Model Evolution Calculus: Semantical View

$$(\text{subsume})\ \frac{\Lambda, K \vdash \Phi, L \vee C}{\Lambda, K \vdash \Phi}\ \text{if } K \geq L$$

The Model Evolution Calculus: Semantical View

$$\text{(resolve)}\ \frac{\Lambda, \overline{K} \vdash \Phi, L \vee C}{\Lambda, \overline{K} \vdash \Phi, C}\ \text{if}\ K \geq L$$

$$\text{(resolve)}\ \frac{\Lambda \vdash \Phi, L \vee C}{\Lambda \vdash \Phi, C}\ \text{if}\ (**)$$

(**) There is a context unifier σ of L with an empty remainder such that $C = C\sigma$

The Model Evolution Calculus: Semantical View

$$(\text{split})\ \frac{\Lambda \vdash \Phi, C \vee L}{\Lambda, L\sigma \vdash \Phi, C \vee L \qquad \Lambda, (\overline{L\sigma})^{sko} \vdash \Phi, C \vee L}\ \text{if } (*)$$

$(*)$ =

1) σ is a context unifier of $(C \vee L)$ against Λ
2) σ is admissible
3) ~~$L\sigma$ is not mixed~~ — No Longer
4) $L\sigma$ not contr. with Λ
5) $(\overline{L\sigma})^{\text{sko}}$ not contr. with Λ

Main Results: Soundness and Completeness

A clause set Φ_0 is unsatisfiable

iff

it has a refutation.

Making ME Efficient

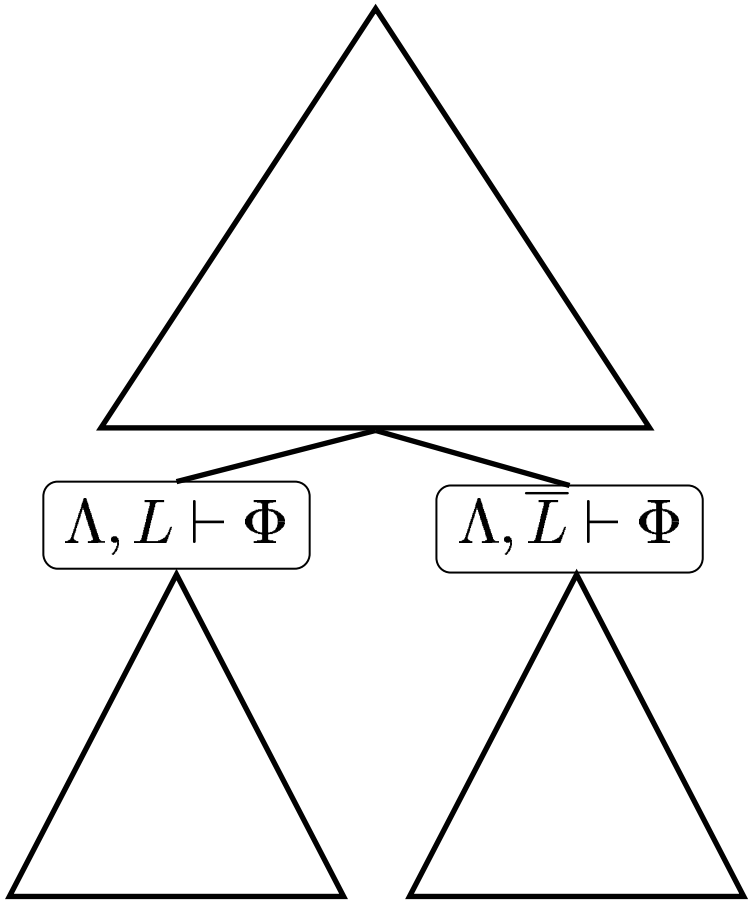
Well-known DPLL improvements:

- **Literal selection** strategies
  Model Elimination:
  can exploit don't care nondeterminism
  for remainer literal to split on

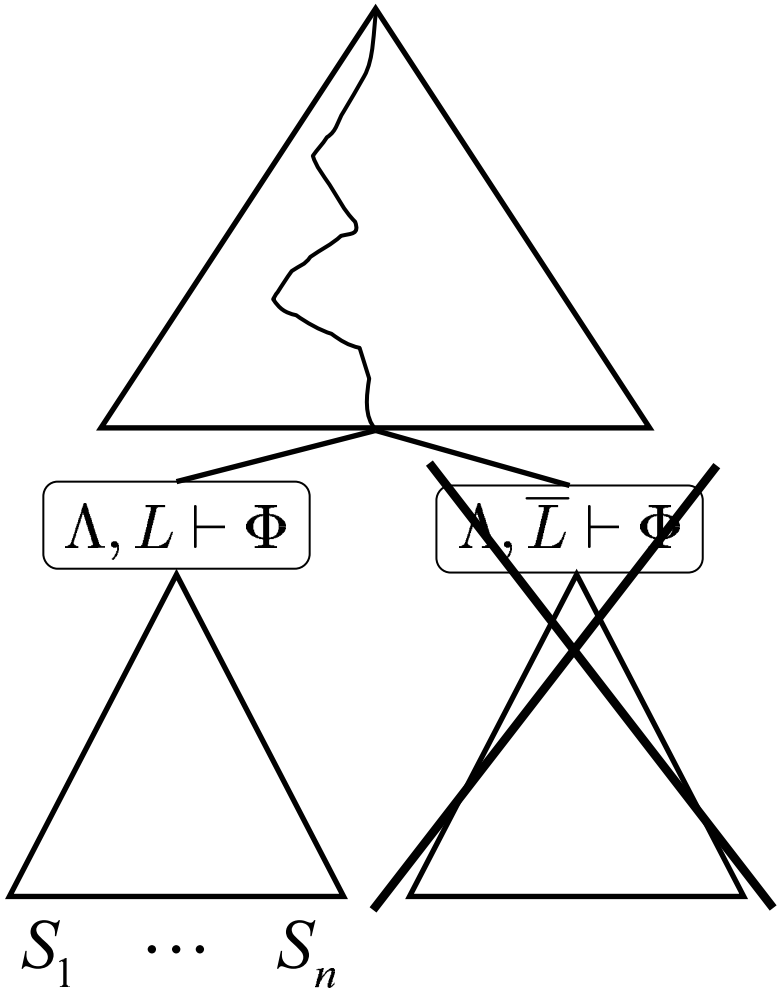
- **Learning** (lemma generation)
  not trivial – future work

- **Intelligent backtracking** (backjumping)

Backjumping



Backjumping



L not used to close left subtree

Implementation

- Darwin is an implementation of the ME.
- It is written in Ocaml^{j}, a variant of ML.
- The algorithm explores derivation trees using iterative deepening on *term depth.*

- Let’s see a demo!

Input file

```
r(a); r(f(a)).                /* r(a) OR r(f(a))                 */
s(X); t(X).                   /* s(X) OR t(X)                    */
p(f(X)); q(f(X)) :- r(a).     /* r(a) -> (p(f(X)) OR q(f(X)))    */
p(f(X)) :- q(f(X)).           /* q(f(X)) -> p(f(X))              */
q(f(X)) :- p(f(X)).           /* p(f(X)) -> q(f(X))              */
false :- q(f(X)), p(f(X)).    /* not (q(f(X)) AND p(f(X)))       */
r(a) :- r(f(a)).              /* r(f(a)) -> r(a)                 */
```

A refutation

[1] Split Left: +r(a)
[2] Split Left: +s(=0)
[3] Split Left: +p(f(=0))
[3] Unit Split: +q(f(=0))
[3] Close:
[3] Backtracking to: [3]
[3] Split Right: -p(f(=0))
[3] Unit Split: +q(f(=0))
[3] Close:
[3] Backtracking to: [1]
[1] Split Right: -r(a)
[1] Assert: -r(f(a))
[1] Close:
END OF DERIVATION

```
r(a); r(f(a)).
s(X); t(X).
p(f(X)); q(f(X)) :- r(a).
p(f(X)) :- q(f(X)).
q(f(X)) :- p(f(X)).
false :- q(f(X)), p(f(X)).
r(a) :- r(f(a)).
```

Summary

- **Full lifting of DPLL achieved**
- **Properties of DPLL preserved**
  - sound and complete
  - proof convergent
  - simplification rules
  - no Commit rule as in FDPLL