# Planning and Optimization

## B4. General Regression, Part II

Malte Helmert and Gabriele Röger

Universität Basel

October 17, 2016

# Regressing Formulas Through Operators

# Regressing Formulas Through Operators: Idea

- We can now regress arbitrary formulas through arbitrary effects.
- The last missing piece is a definition of regression through operators, describing exactly in which states $s$ applying a given operator $o$ leads to a state satisfying a given formula $\varphi$.
- There are two requirements:
  - The operator $o$ must be applicable in the state $s$.
  - The resulting state $s[\![o]\!]$ must satisfy $\varphi$.

# Regressing Formulas Through Operators: Definition

**Definition (Regressing a Formula Through an Operator)**

Let $o$ be an operator, and let $\varphi$ be a formula over state variables.

The regression of $\varphi$ through $o$, written $regr_o(\varphi)$, is defined as the following logical formula:

$$regr_o(\varphi) = pre(o) \wedge regr_{eff(o)}(\varphi).$$

# Regressing Formulas Through Operators: Correctness (1)

**Theorem (Correctness of $regr_o(\varphi)$)**

*Let $\varphi$ be a logical formula, $o$ an operator and $s$ a state.*

*Then $s \models regr_o(\varphi)$ iff $o$ is applicable in $s$ and $s[\![o]\!] \models \varphi$.*

# Regressing Formulas Through Operators: Correctness (2)

Reminder: $regr_o(\varphi) = pre(o) \wedge regr_{eff(o)}(\varphi)$

**Proof.**

Case 1: $s \models pre(o)$.

Then $o$ is applicable in $s$ and the statement we must prove simplifies to: $s \models regr_{eff(o)}(\varphi)$ iff $s[\![o]\!] \models \varphi$.
This was proved in the previous lemma.

# Regressing Formulas Through Operators: Correctness (2)

Reminder: $regr_o(\varphi) = pre(o) \wedge regr_{eff(o)}(\varphi)$

**Proof.**

Case 1: $s \models pre(o)$.

Then $o$ is applicable in $s$ and the statement we must prove simplifies to: $s \models regr_{eff(o)}(\varphi)$ iff $s[\![o]\!] \models \varphi$.
This was proved in the previous lemma.

Case 2: $s \not\models pre(o)$.

Then $s \not\models regr_o(\varphi)$ and $o$ is not applicable in $s$.
Hence both statements are false and therefore equivalent. □

# Regression Examples (1)

## Examples: compute regression and simplify to DNF

- $regr_{\langle a,b\rangle}(b)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot))$
  $\equiv a$
- $regr_{\langle a,b\rangle}(b \wedge c \wedge d)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot)) \wedge (\bot \vee (c \wedge \neg\bot)) \wedge (\bot \vee (d \wedge \neg\bot))$
  $\equiv a \wedge c \wedge d$
- $regr_{\langle a,b\wedge c\rangle}(b \wedge \neg c)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot)) \wedge \neg(\top \vee (c \wedge \neg\bot))$
  $\equiv a \wedge \top \wedge \bot$
  $\equiv \bot$

# Regression Examples (2)

## Examples: compute regression and simplify to DNF

- $regr_{\langle a, c \rhd b \rangle}(b)$
  $\equiv a \wedge (c \vee (b \wedge \neg \bot))$
  $\equiv a \wedge (c \vee b)$
  $\equiv (a \wedge c) \vee (a \wedge b)$
- $regr_{\langle a, (c \rhd b) \wedge ((d \wedge \neg c) \rhd \neg b) \rangle}(b)$
  $\equiv a \wedge (c \vee (b \wedge \neg (d \wedge \neg c)))$
  $\equiv a \wedge (c \vee (b \wedge (\neg d \vee c)))$
  $\equiv a \wedge (c \vee (b \wedge \neg d) \vee (b \wedge c))$
  $\equiv a \wedge (c \vee (b \wedge \neg d))$
  $\equiv (a \wedge c) \vee (a \wedge b \wedge \neg d)$

# Practical Issues

# Emptiness and Subsumption Testing

The following two tests are useful when performing regression searches to avoid exploring unpromising branches:

- Test that $regr_o(\varphi)$ does not represent the empty set (which would mean that search is in a dead end).
  For example, $regr_{\langle a, \neg p \rangle}(p) \equiv a \wedge (\bot \vee (p \wedge \neg \top)) \equiv \bot$.
- Test that $regr_o(\varphi)$ does not represent a subset of $\varphi$ (which would mean that the resulting search state is worse than $\varphi$ and can be pruned).
  For example, $regr_{\langle b, c \rangle}(a) \equiv a \wedge b$.

Both of these problems are NP-complete.

# Formula Growth

The formula $regr_{o_n}(\ldots regr_{o_2}(regr_{o_1}(\varphi)))$ may have size $O(|\varphi||o_1||o_2|\ldots|o_{n-1}||o_n|)$, i.e., the product of the sizes of $\varphi$ and the operators.

$\rightsquigarrow$ worst-case exponential size $\Omega(|\varphi|^n)$

## Logical Simplifications

- $\bot \wedge \varphi \equiv \bot$, $\top \wedge \varphi \equiv \varphi$, $\bot \vee \varphi \equiv \varphi$, $\top \vee \varphi \equiv \top$
- $a \vee \varphi \equiv a \vee \varphi[\bot/a]$, $\neg a \vee \varphi \equiv \neg a \vee \varphi[\top/a]$, $a \wedge \varphi \equiv a \wedge \varphi[\top/a]$, $\neg a \wedge \varphi \equiv \neg a \wedge \varphi[\bot/a]$
- idempotence, absorption, commutativity, associativity, ...

# Restricting Formula Growth in Search Trees

**Problem** very big formulas obtained by regression

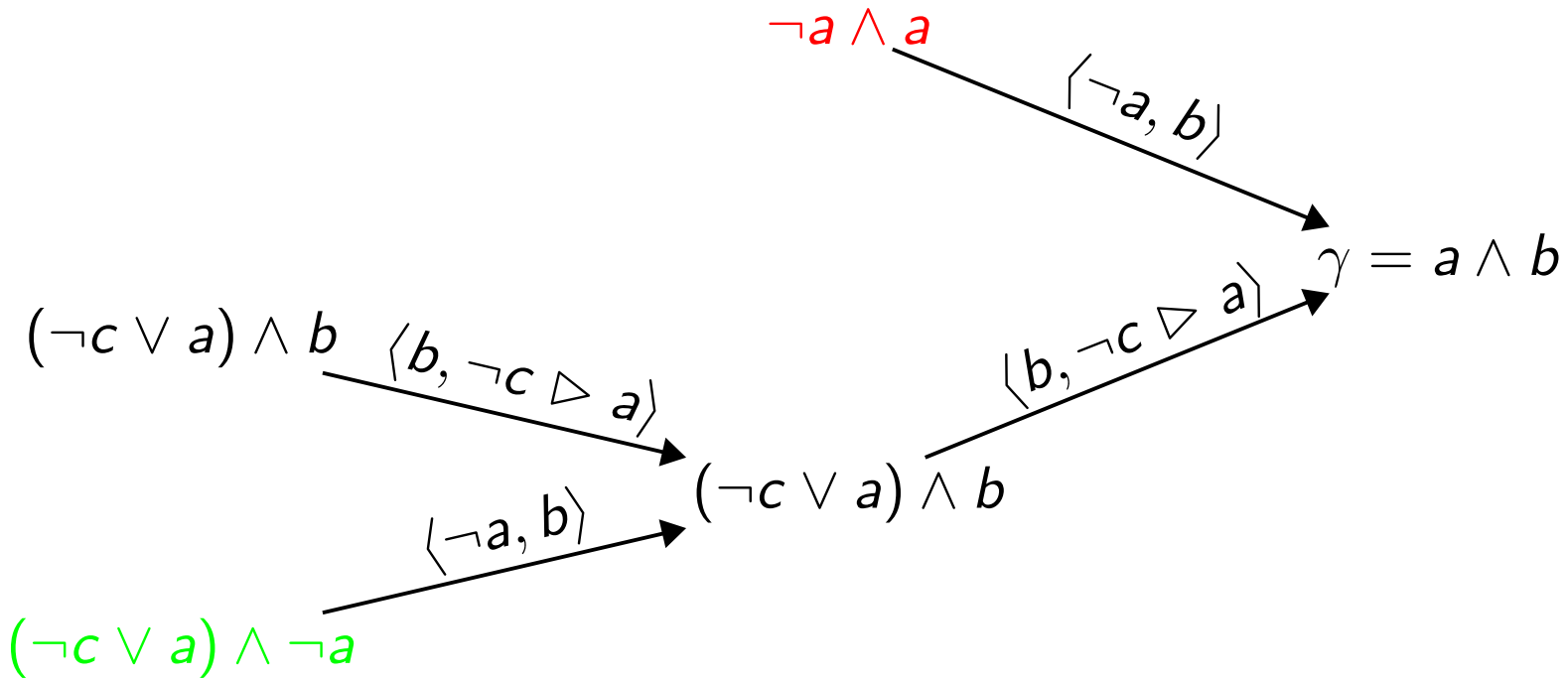
**Cause** disjunctivity in the (NNF) formulas
(formulas without disjunctions easily convertible
to monomials $\ell_1 \wedge \cdots \wedge \ell_n$ where $\ell_i$ are literals
and $n$ is at most the number of state variables)

**Idea** split disjunctive formulas when generating search trees

# Unrestricted Regression: Search Tree Example

Unrestricted regression: do not treat disjunctions specially

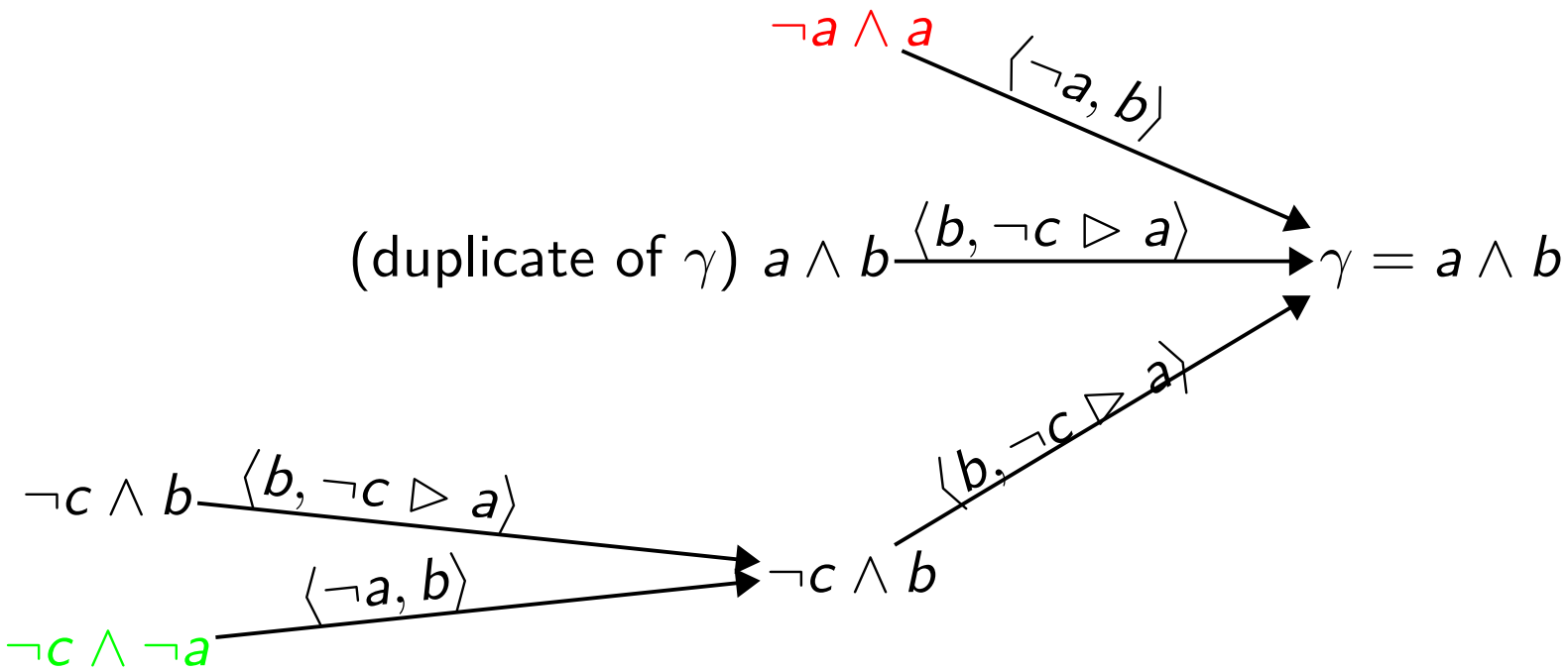
Goal $\gamma = a \wedge b$, initial state $I = \{a \mapsto \mathbf{F}, b \mapsto \mathbf{F}, c \mapsto \mathbf{F}\}$.



# Full Splitting: Search Tree Example

Full splitting: always split all disjunctive formulas

Goal $\gamma = a \wedge b$, initial state $I = \{a \mapsto \mathbf{F}, b \mapsto \mathbf{F}, c \mapsto \mathbf{F}\}$.
$(\neg c \vee a) \wedge b$ in DNF: $(\neg c \wedge b) \vee (a \wedge b)$
$\rightsquigarrow$ split into $\neg c \wedge b$ and $a \wedge b$



# General Splitting Strategies

Alternatives:

1. Do nothing (unrestricted regression).
2. Always eliminate all disjunctivity (full splitting).
3. Reduce disjunctivity if formula becomes too big.

Discussion:

- With unrestricted regression formulas may have sizes that are exponential in the number of state variables.
- With full splitting search tree can be exponentially bigger than without splitting.
- The third option lies between these two extremes.

# Summary

# Summary

- Regressing a formula $\varphi$ through an operator involves regressing $\varphi$ through the effect and enforcing the precondition.
- When applying regression in practice, additional considerations come into play, including:
  - emptiness testing to prune dead-end search states
  - subsumption testing to prune dominated search states
  - logical simplifications and splitting to restrict formula growth