Satisfiability Modulo Theories

Motivation, Process, Solvers

Yu-Yun Dai

Automatic Verification, Spring 2012

Outline

- Introduction
  - Motivation of SMT
  - First Order Logic
- Theories of Interest
- SMT competition
- Eager approach
  - Algorithm and Solving procedure
  - Solvers: boolector, Yices 1.0
- Lazy approach
  - DPLL(T)
  - Solvers: MATHSAT5, Z3

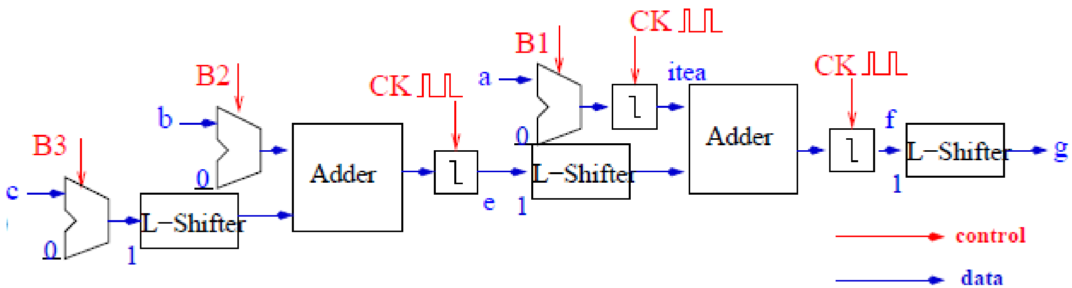
Examples for SMT problems(1)

- Planning with Resources
- Straightforward to encode into SMT(LA(Q))

Example:	
(Deliver)	// goal
∧ (MaxLoad)	// load constraint
∧ (MaxFuel)	// fuel constraint
∧ (Move → MinFuel)	// move requires fuel
∧ (Move → Deliver)	// move implies delivery
∧ (GoodTrip →Deliver)	// a good trip requires
∧ (GoodTrip → AllLoaded)	// a full delivery
∧ (MaxLoad → (load 30))	// load limit
∧ (MaxFuel → (fuel 15))	// fuel limit
∧ (MinFuel → (fuel 7 + 0.5load))	// fuel constraint
∧ (AllLoaded → (load = 45))	// more than MaxLoad......

Examples for SMT problems(2)

- Verification of HW circuit designs & microcode



- *Control paths* handled by Boolean reasoning
- *Data paths* information abstracted into theory-specific terms
  - *words* (bit-vectors, integers, EUF vars, ...): $a[31:0]$, a
  - *word operations*: (BV, EUF, AR, LA(Z), NLA(Z) operators)

    $x[15:0] = (y[15:8] :: z[7:0]) << w[3:0]$,

    $(a = a_L + 2^{16} a_H)$, $(m_1 = store(m_0, l_0, v_0))$, ...
- SMT on BV, EUF, AR, modulo-LA(Z) required

Introduction – why SMT?

- SAT solvers are developed very well.
  - SAT has benefited many areas: AI, formal methods
- However……
  - applications in these fields require determining the satisfiability of formulas in more expressive logics such as **first-order logic**
  - Bit-level encoding (bit-blasting) usually exploit problem-specific structures makes hardware verification not scalable
    - (the example for bit-blasting is in Eager approach)
- General first-order satisfiability is Undecidable.
  - It is only semi-decidable.
  - general-purpose first-order theorem provers are typically not able to solve such formulas directly

Introduction – why SMT? (cont.)

- In most applications…
  - Not require general first-order satisfiability
  - fixed interpretations of certain predicate and function symbols
- Can we solve the simpler formulae directly?

- Can we adopt the wisdom of SAT solvers?
  - DPLL, non-chronological backtracking, conflict-driven learning, two-literal watch scheme, VSIDS
- Can we make SAT solvers structure-aware?
- So……here comes SMT !

Introduction- First Order Logic (1)

- Syntax : First-Order Languages consist of
  - Logical symbols
    - variables : $x, y, z, \ldots$
    - logic operators and quantifiers : $\neg \vee \wedge \rightarrow, \exists \forall$
    - equality symbol: = (optional)
  - Parameters
    - constant symbols : $c_1, c_2, \ldots$ (countable)
    - function symbols : $f, g, \ldots$ (possibly empty)
    - predicate symbols : $p, q, \ldots$ (possibly empty)
  - Ex. $\Sigma_{\mathbb{N}} = \{ \{0\}, \{S, +\}, \{=\} \}$
- To specify a language, we need to specify
  - Presence of "="
  - Symbols

Introduction- First Order Logic(2)

- Terms
  - Every constant c_1 or variable x is a term.
  - If $t_1, \ldots, t_k$ are terms and f is a k-ary function symbol, $f(t_1, \ldots, t_k)$ is a term.
  - Ex: $SS0$
- Formula
  - *True* and *False* are atomic formulas.
  - If $t_1, \ldots, t_k$ are terms and P is a k-ary predicate symbol, $P(t_1, \ldots, t_k)$ is an atomic formula.
  - Ex: $< x\ y$ (define $<$ as predicate sumbol)
- Well Form Formulae:
  - expression built up from atomic formulas by applying these operations: $\neg \vee \wedge \rightarrow, \exists \forall$
  - Ex: $(x < y) \vee (x = y)$
- Free variable:
  - variables in a formula are those not bound by a quantifier
- Sentence :
  - Formula without free variable

Introduction- First Order Logic(3)

- Sematic : **Structure** $\mathcal{A}$ consists of
  - Universe (or domain) of $\mathcal{A}$
  - Interpretation for each parameter
    - (constant, function, predicate)
  - Ex. A $\Sigma_{\mathbb{N}}$-structure
  - $\{ \{0\}, \{S^{\mathcal{A}} := succ, +^{\mathcal{A}} := plus, =^{\mathcal{A}} := equal\} \}$
- Define a structure $\mathcal{A}$ satisfies a wff ϕ with assignment **s**
  - The translation of ϕ determined by A is true, where variable x is translated as s(x) wherever it occurs free.
- $\mathcal{A}$ satisfies ϕ with <u>every</u> **s** :
  - ϕ is true in $\mathcal{A}$
  - $\mathcal{A}$ is a model of ϕ

Introduction- First Order Logic(4)

- A **theory** $\mathcal{T}$ (over a structure)
  - a set of first-order sentences closed under logical implication.
- $\mathcal{A}$ is a **model** for the theory $\mathcal{T}$
  - if all sentences of $\mathcal{T}$ are true in $\mathcal{A}$.
- So far, that is the definition from the book
  - **"A Mathematical Introduction to Logic"**

SATISFIABILITY OF SAT AND SMT

- Satisfiability is the problem of determining if a formula has a *model*
  - *Model* :**structure with variable assignment**.
- In purely Boolean cases
  - a model is a truth assignment to the Boolean variables.
- In first-order cases
  - a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.
- A formula F is satisfiable if there is an interpretation (model)M such that
  - $M \models F$.
  - Otherwise, the formula F is unsatisfiable.

Outline

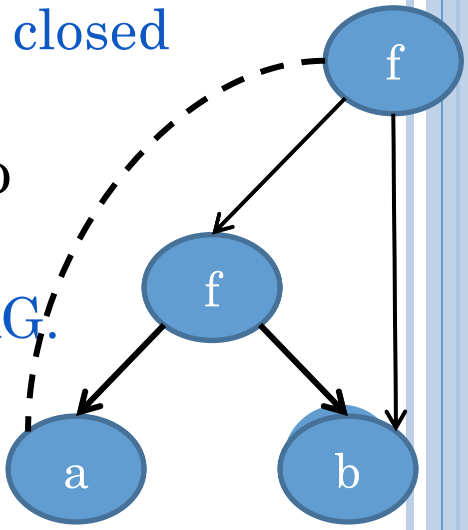
- Introduction
  - Motivation of SMT
  - First Order Logic
- Theories of Interest
  - Theory of equality T_E
  - Theory of Reals T_R
  - Theory of Integers T_Z
  - Theory of Arrays AR
  - Theory of Bitvectors BV
- SMT Competition
- Eager approach
  - Algorithm and Solving procedure
  - Solvers: boolector, Yices 1.0
- Lazy approach
  - DPLL(T)
  - Solvers: MATHSAT5, Z3

Theory of equality $\mathcal{T}_{\mathbb{E}}$

- Theory of equality and uninterpreted functions.
- $\Sigma_{\mathbb{E}} = \{ \{c_1, \ldots\}, \{f_1, \ldots\}, \{=\} \}$
  - Ex. $[f(f(\mathrm{a})) = \mathrm{a}] \wedge \left[f\left(f(f(\mathrm{a}))\right) = \mathrm{a}\right] \wedge [f(a) \neq a]$
    - $\mathcal{T}_{\mathbb{E}}$-unsatisfiable
- Axiom schema
  - $\forall x. (x = x)$ (reflexivity)
  - $\forall x, y. (x = y \rightarrow y = x)$ (symmetry)
  - $\forall x, y, z. (x = y \wedge y = z \rightarrow x = z)$ (transitivity)
  - $\forall \vec{x}, \vec{y}. (\bigwedge x_i = y_i \rightarrow f(\vec{x}) = f(\vec{y}))$ (congruence)
- The satisfiability problem for conjunction of literals in $\mathcal{T}_{\mathbb{E}}$ is decidable in polynomial time using congruence closure.

Congruence closure (1)

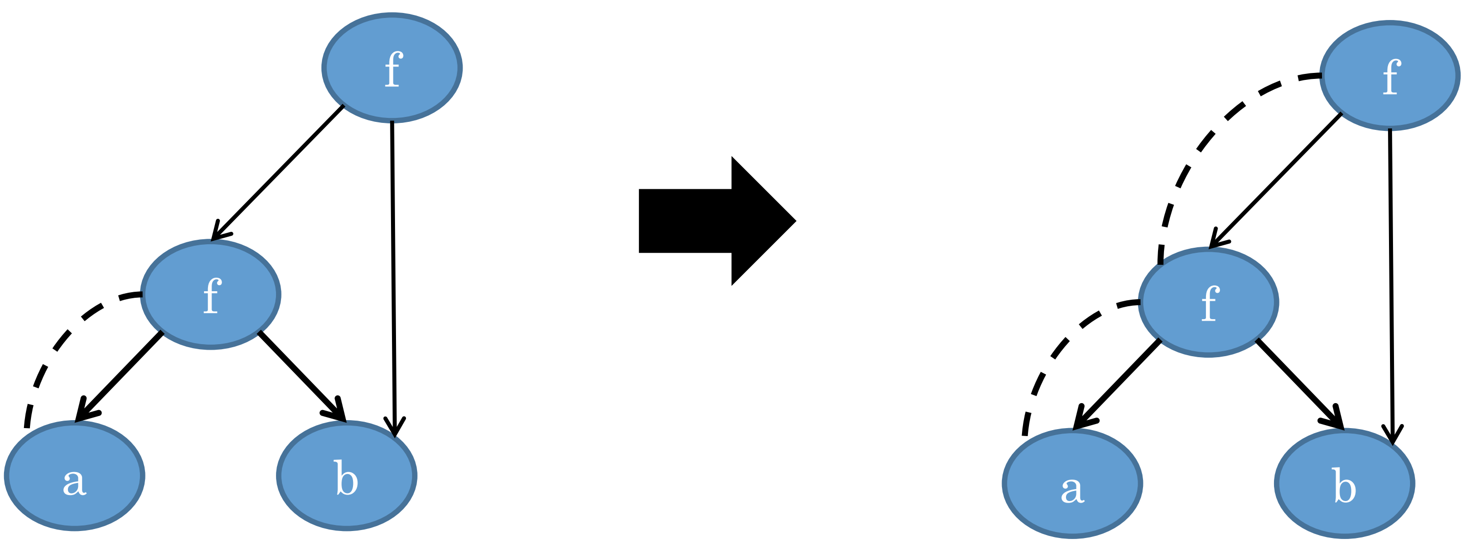
- Given binary relation R over S.
- The equivalence closure of R
  - The unique minimal extension R′ of R, that is closed under equivalence relation
    - reflexivity, symmetry, transitivity.
- congruence closure of R
  - The unique minimal extension R′ of R, that is closed under congruence relation.
- We use the directed acyclic graph (DAG) to represent terms:
  - A term corresponds to exactly one node in DAG.
  - Equalities are represented as dot lines.
- Ex: f (f (a, b), b) = a



Congruence closure (2)

- Computing congruence closure:
  - Pick arbitrary representatives for all equivalence classes (nodes connected by dotted edges)
  - Construct congruence closure for these edges.

  - Ex: $f(a, b) = a \Rightarrow f(f(a, b), b) = f(a, b)$



Real Linear Arithmetic $\mathcal{T}_{\mathbb{Q}}$

- $\Sigma_{\mathbb{Q}} = \{ \{\mathbb{Z}\}, \{+, -\}, \{<, =\} \}$,
  $A_{\mathbb{Q}}$ = the set of rational numbers
  - Ex. $(2x < 4) \wedge (2x > 2)$ ($\mathcal{T}_{\mathbb{Q}}$-satisfiable)
  - no need to consider irrational under linear real arithmetic.
- SAT($\mathcal{T}_{\mathbb{Q}}$) can be solved by polynomial time algorithm.
  - Fourier-Motzkin variable elimination algorithm.
  - Simplex algorithm
    - exponential methods
    - tend to perform best in practice.

Quantifier Elimination

- If a formula with no free and no quantifiers, then it is easy to determine its truth value
  - $10 > 11 \vee 3 + 4 < 5 \times 3 - 6$
- Quantifier elimination
  - take input P with n quantifiers
  - turn it into equivalent formula P′ with m quantifiers, where m < n.
- Eventually $P \equiv P' \equiv \cdots \equiv Q$ and Q has no quantifiers.
- Q will be trivially true or false, and that is the decision.

Fourier-Motzkin Theorems

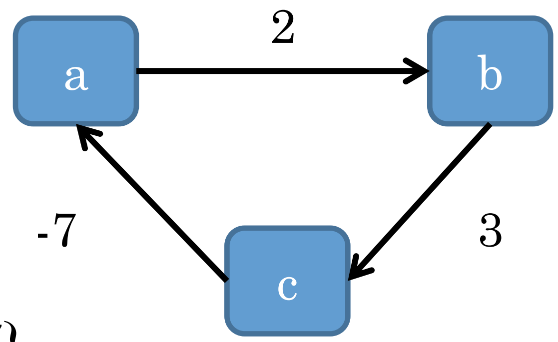
- The following simple facts are the basis for a very simple quantifier elimination procedure.
- transitivity.
  - $(x < y \land y \leq z) \Rightarrow x < z.$
- Over **R**, with a, b > 0:
  - $\exists x.(c \leq ax \land bx \leq d) \equiv (bc \leq ad)$
  - $\exists x.(c < ax \land bx \leq d) \equiv \exists x.(c \leq ax \land bx < d)$
    $\equiv \exists x.(c < ax \land bx < d) \equiv (bc < ad)$
- Proof:
  - For $bc < ad \Rightarrow (\exists x.c < ax \land bx \leq d)$
    - take x to be d/b ➔ $c < a(d/b)$ and $b(d/b) \leq d$.
- Combining Many Constraints
  - $\exists x.(c \leq ax \land b_1x \leq d_1 \land b_2x \leq d_2)$

  $\equiv b_1c \leq ad_1 \land b_2c \leq ad_2$

Difference Logic

- Difference logic is a fragment of linear arithmetic.
- Atoms have the form:
  - $x - y \leq c$.
- Most linear arithmetic atoms found in hardware and software verification are in this fragment.
- The quantifier free satisfiability problem is solvable in O(VE).
  - V: number of variables
  - E: number of Atoms
- (solve by Bellman-Ford algorithm)

Difference Logic (2)



- Given M = {a−b ≤ 2, b−c ≤ 3, c−a ≤ −7},
  - construct weighted graph G(M)
  - M is T-inconsistent iff G(M) has a negative cycle
- Any negative cycle $a1 \xrightarrow{k1} a2 \xrightarrow{k2} a3 \rightarrow \ldots \rightarrow an \xrightarrow{kn} a1$ corresponds to a set of literals:
  - a1 − a2 ≤ k1
  - a2 − a3 ≤ k2
  - . . .
  - an − a1 ≤ kn
- If we add them all, we get 0 ≤ k1 + k2 + . . . + kn
  - negative cycle implies k1 + k2 + . . . + kn < 0 ➔ inconsistent

Integer Linear Arithmetic $\mathcal{T}_{\mathbb{Z}}$

- $\Sigma_{\mathbb{Z}} = \{ \{\mathbb{Z}\}, \{+, -\}, \{<, =\} \}$

  $A_{\mathbb{Z}}$= the set of integers
  - Ex: $(2x < 4) \wedge (2x > 2)$ ($\mathcal{T}_{\mathbb{Z}}$-unsatisfiable)
- SAT($\mathcal{T}_{\mathbb{Z}}$) is NP-complete.
  - Fourier-Motzkin algorithm doesn't work well.
- However, it becomes undecidable if multiplication is introduced in $\mathcal{T}_{\mathbb{Z}}$.
  - Ex: $x \times y<5$

Theory of Arrays $\mathcal{T}_{AR}$

- The theory of arrays ($\mathcal{T}_{AR}$) aims at modeling the behavior of arrays/memories.
  - write(a, i , v) ; read(a, i)
  - a: array, i: index, v: element
- Axiom schema
  - McCarthy's axioms
    - $\forall a, i, v. read(write(a, i, v), i) = v$
    - $\forall a, i, j, v. (i \neq j) \rightarrow [read(write(a, i, v), i) = read(a, j))]$
  - Extensionality axioms
    - $\forall a, b. (\forall i. (read(a, i) = read(b, i))) \rightarrow (a = b)$
- SAT($\mathcal{T}_{\mathbb{A}}$) is NP-complete modulo $\mathcal{T}_{\text{elem}}$.

Theory of Bitvectors $\mathcal{T}_{bv}$

- Domains : vectors of bits.
  - a[7:0]
- Like hardware design
- Operators:
  - read, write: like array
  - extraction, concatenation:
    - a[7:0]; b[3:0] = a[3:0]; c = { a, b}
  - bit-wise operations
    - &, |, ^
  - arithmetic operations
    - +, -, *, /, %
- SAT($\mathcal{T}_{bv}$) is NP-complete.

OUTLINE

- Introduction
  - Motivation of SMT
  - First Order Logic
- Theories of Interest
- **SMT competition**
- Eager approach
  - Algorithm and Solving procedure
  - Solvers: boolector, Yices 1.0
- Lazy approach
  - DPLL(T)
  - Solvers: MATHSAT5, Z3

History of SMT competition

- Since 2005
- 2005&2006: Only several quantifier free linear arithmetic categories
  - Outperform solvers: Yices 0.1, MathSAT3
- 2007: BV problems added
  - Z3 0.1, Yices 1.0, MathSAT4
- 2008:
  - Z3.2 dominated most categories,
  - Boolector won in BV categories
  - Poor MathSAT4.2 and Yices2……

Theories in SMT Competition

- QF_UF : equality and uniterpreted functions.
- QF_RDL/QF_IDL : real/integer difference logic.
- QF_LIA/QF_LRA : linear real/integer arithmetic
- QF_NIA : nonlinear integer arithmetic
- QF_AX : arrays with extensionality.
- QF_BV : bit-vectors.
- AUFLIRA : arrays, UF, LIA, LRA
- AUFNIRA : arrays, UF, NIA, NRA
- All of the above are decidable!

History of SMT competition

- 2009: more and more categories…….
  - MathSAT 4.3, Yices2.0 outperformed others in most categories
  - Boolector only took BV domain(still worked well)
  - Z3 was in summer vacation?
- 2010: first year of parallel track
  - Many new solvers appeared
  - Z3 and Boolector took summer vacation again…..
  - MathSAT5, CVC3, openSMT
- 2011
  - Z3 kicked other solvers…….
  - MathSAT5, CVC3, openSMT

What we know from the history?

- If we focus on BV problems:
  - Why boolector works so well?
- What's going on with Yices?
- How can MathSAT and Z3 outperform others?

- Case-dependent? Luck?

Outline

- Introduction
  - Motivation of SMT
  - First Order Logic
- Theories of Interest
- SMT competition
- Eager approach
  - Algorithm and Solving procedure
  - Solvers: boolector, Yices 1.0
- Lazy approach
  - DPLL(T)
  - Solvers: MATHSAT5, Z3

General idea of Eager approach

- Translate the original formula to an satisfiability-equivalent Boolean formula in a single step.
  - Boolector, BV part in Yices
  - The performance is related to the size of SAT instance.
- Bit-blasting example:
  - x[3:0] + y[3:0] = z[3:0]
  - it might introduce other variables
  - ➔z[0] = x[0]⊕y[0], c[0] = x[0]y[0], z[1] = x[1] ⊕y[1] ⊕c[0].....
- Smaller domain encoding
  - Convert F_{orig} to F_{arith}
  - Replace each constraint in F_{arith} with a fresh Boolean variables to get a Boolean formulaF_{var}
  - Convert F_{arith} to Boolean formula F_{bool}.
  - Ex: (x[3:0] + y[3:0] = z[3:0]) ∨ (w[3:0] = 7) = > A ∨ B

Example for small encoding

- over Integer Linear Arithmetic $\mathcal{T}_{\mathbb{Z}}$

F_{arith} $\Phi =$

$(\ (x+y < 5)\ \lor \neg(x+y > 10)\ )$

$\land(\ (x+y < 5)\ \lor \neg(x\text{-}y=3)\ )$

$\land(\ (x+y > 10) \lor\ \ (x\text{-}y=3)\ )$

$\land\ (\ \neg(x+y < 5))$

- F_{var} $\Phi' =$

$(A \lor \neg B)$

$\land(A \lor \neg C)$

$\land(B \lor\ \ C)$

$\land\ (\neg A)$

After smaller domain encoding

- If UNSAT, we can return the answer.
- However, we might miss some conflicts under smaller encoding
- Ex: [$\neg(x + y = 3) \vee (x + y < 2)$] $\wedge$ [$(x + y = 3)$]
  - After smaller encoding: [$\neg A \vee B$] $\wedge$ [A]
  - Assign $A = 1$, $B = 1$ ➔ SAT!
  - However……$(x + y = 3) \wedge (x + y < 2)$ ➔ UNSAT!
- Worse case, we still need bit-blasting!
- Why Boolector is so powerful?

The secret of Boolector-rewrite

- Example can not be handled by small encoding
  - $(x+y = p) \wedge (p+x = q) \wedge (2x = r) \wedge (r+y = s) \wedge \neg (q=s)$
- Boolector contains crazy, rule-based rewrite!
  - Commutative property
  - Associativity
  - Symmetry
- Better encoding for special operators:
  - Ex: Shift operator
    - c[3:0] = a[3:0] << b[1:0]
    - b[1] → c[3:0] ≥ a[3:0]*2

Resource of Boolector

- Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria.
- Open source: http://fmv.jku.at/boolector/
  - Picosat needed
  - http://fmv.jku.at/boolector/README
  - The version for smtlib2 does not be uploaded
- We can use the simpler format BTOR to write input cases☺
  - BTOR example

```
1 var 6
2 var 6
3 var 6
4 add 6 1 2
5 add 6 4 3
6 add 6 2 3
7 add 6 6 1
8 eq 1 5 7
9 root 1 8
```

Resource of YICES

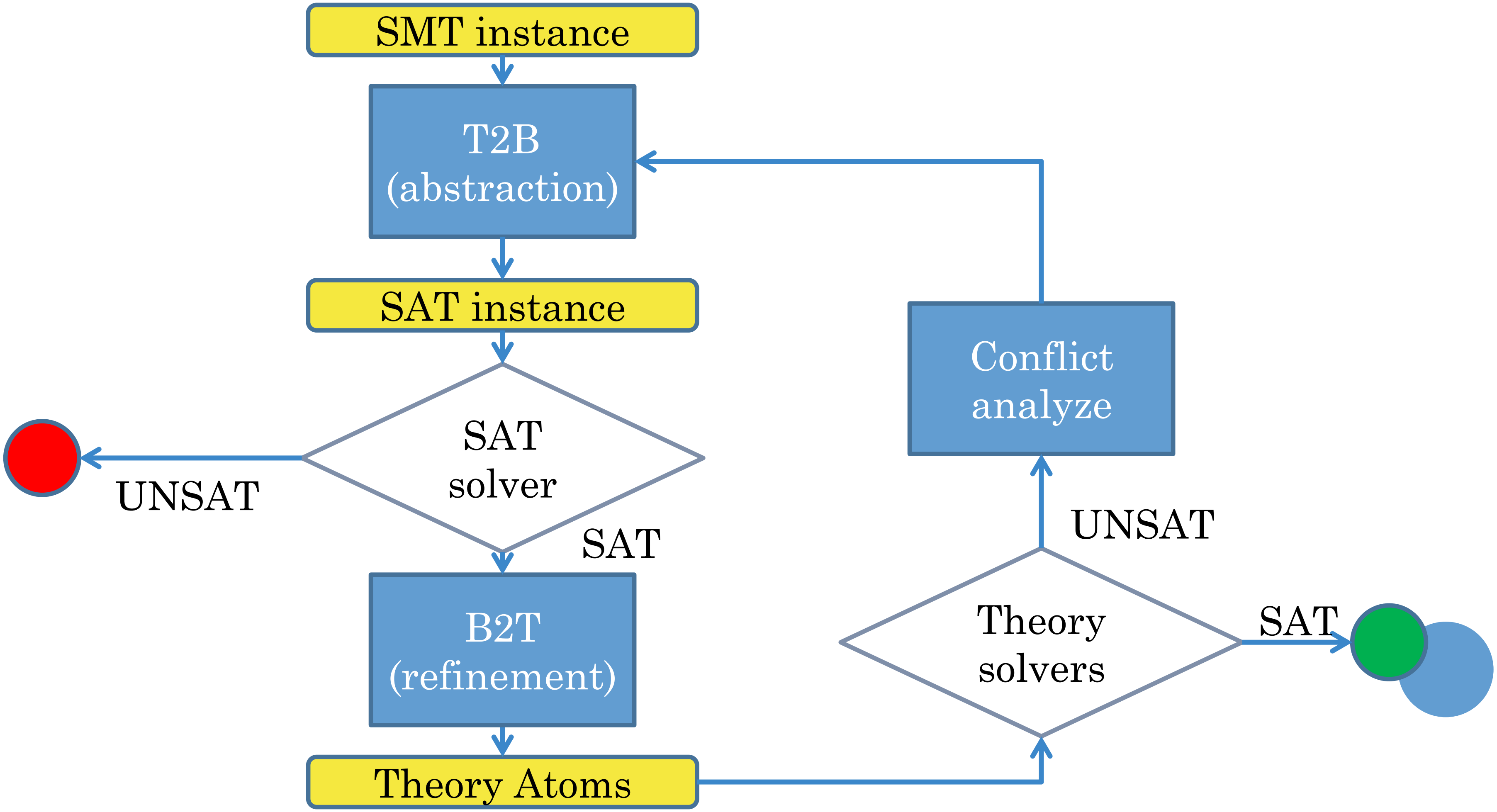
- Computer Science Laboratory, SRI International Menlo Park, CA
- http://yices.csl.sri.com/
- http://yices-wiki.csl.sri.com/index.php/Main_Page
- For BV, only some simplification rule, and bit-blasting!
- For other theories, apply lazy approach
- No source code
- Read SMT-LIB format and its own format

Outline

- Introduction
  - Motivation of SMT
  - First Order Logic
- Theories of Interest
- SMT competition
- Eager approach
  - Algorithm and Solving procedure
  - Solvers: boolector, Yices 1.0
- Lazy approach
  - DPLL(T)
  - Solvers: MATHSAT5, Z3

Overview of Lazy approach

- Combine SAT and Theory Solvers



DPLL($\mathcal{T}$): T2B (Theory-to-Boolean)

- T 2B (Theory-to-Boolean)
  - a bijective function,
  - maps Boolean atoms into themselves
  - non-Boolean $\mathcal{T}$-atoms into fresh Boolean atoms
  - Two atom instances are mapped into the same Boolean atom iff they are syntactically identical.
- B2T := T2B^{-1} (Boolean-to-Theory)
  - T2B and B2T are also called Boolean abstraction and Boolean refinement respectively.

Example of T2B

- $\phi := \{ \neg(2x_2 - x_3 > 2) \lor A_1\}$
  $\land \{ \neg A_2 \lor (x_1 - x_5 \leq 1) \}$
  $\land \{ (3x_1 - 2x_2 \leq 3) \lor A_2\}$
  $\land \{ \neg(2x_3 + x_4 \geq 5) \lor \neg(3x_1 - x_3 \leq 6) \lor \neg A_1\}$
  $\land \{ A_1 \lor (3x_1 - 2x_2 \leq 3) \}$
  $\land \{ (x_2 - x_4 \leq 6) \lor (x_5 = 5 - 3x_4) \lor \neg A_1\}$
  $\land \{ A_1 \lor (x_3 = 3x_5 + 4) \lor A_2\}$
- $T\,2B(\phi) := \{\neg B_1 \lor A_1\}$
  $\land \{\neg A_2 \lor B_2\}$
  $\land \{B_3 \lor A_2\}$
  $\land \{\neg B_4 \lor \neg B_5 \lor \neg A_1\}$
  $\land \{A_1 \lor B_3\}$
  $\land \{B_6 \lor B_7 \lor \neg A_1\}$
  $\land \{A_1 \lor B_8 \lor A_2\}$

Integration between SAT solver and theory solvers

- Lazy Integration
  - Theory solvers are triggered only after SAT solver determines all variables
  - Return learning clauses after conflicts occur
  - Easier to implement
- Eager Integration
  - theory solver participates in early stages
    - value propagation (implications)
    - conflict analysis
  - Find the conflict sources earlier
  - Require much more implementation works

Example for integration

- Input instance:
  - $[A_1 \vee (u - w \leq 5)]$
  $\wedge[A_2 \vee (v + w \leq 6)]$
  $\wedge[A_3 \vee (z = 0)]$
  $\wedge[A_4 \vee (u + v \geq 12)]$
  $\wedge[\neg A_3 \vee \neg A_4]$
  $\wedge[(x = z + 1) \vee (x = z + 3) \vee (x = z + 5) \vee (x = z + 7)]$
  $\wedge[(y = z + 2) \vee (y = z + 4) \vee (y = z + 6)]$
  $\wedge[(u + v - 4x - 4y = 0)]$
- After T2B
  - $[A_1 \vee B_1]$
  $\wedge[A_2 \vee B_2]$
  $\wedge[A_3 \vee B_3]$
  $\wedge[A_4 \vee B_4]$
  $\wedge[\neg A_3 \vee \neg A_4]$
  $\wedge[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$
  $\wedge[B_{71} \vee B_{72} \vee B_{73}]$
  $\wedge[1]$

Lazy Integration

SAT instance:

$[A_1 \vee B_1]$
$[A_2 \vee B_2]$
$[A_3 \vee B_3]$
$[A_4 \vee B_4]$
$[\neg A_3 \vee \neg A_4]$
$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$
$[B_{71} \vee B_{72} \vee B_{73}]$
$[1]$

SAT solver $\rightarrow$

$\neg A_1 \rightarrow B_1$
$\neg A_2 \rightarrow B_2$
$\neg A_3 \rightarrow B_3$
$\neg A_4 \rightarrow B_4$
B_{61}
B_{71}

SAT!! Go to theory solver!

Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$

In the integer solver……

B_1

B_2

B_3

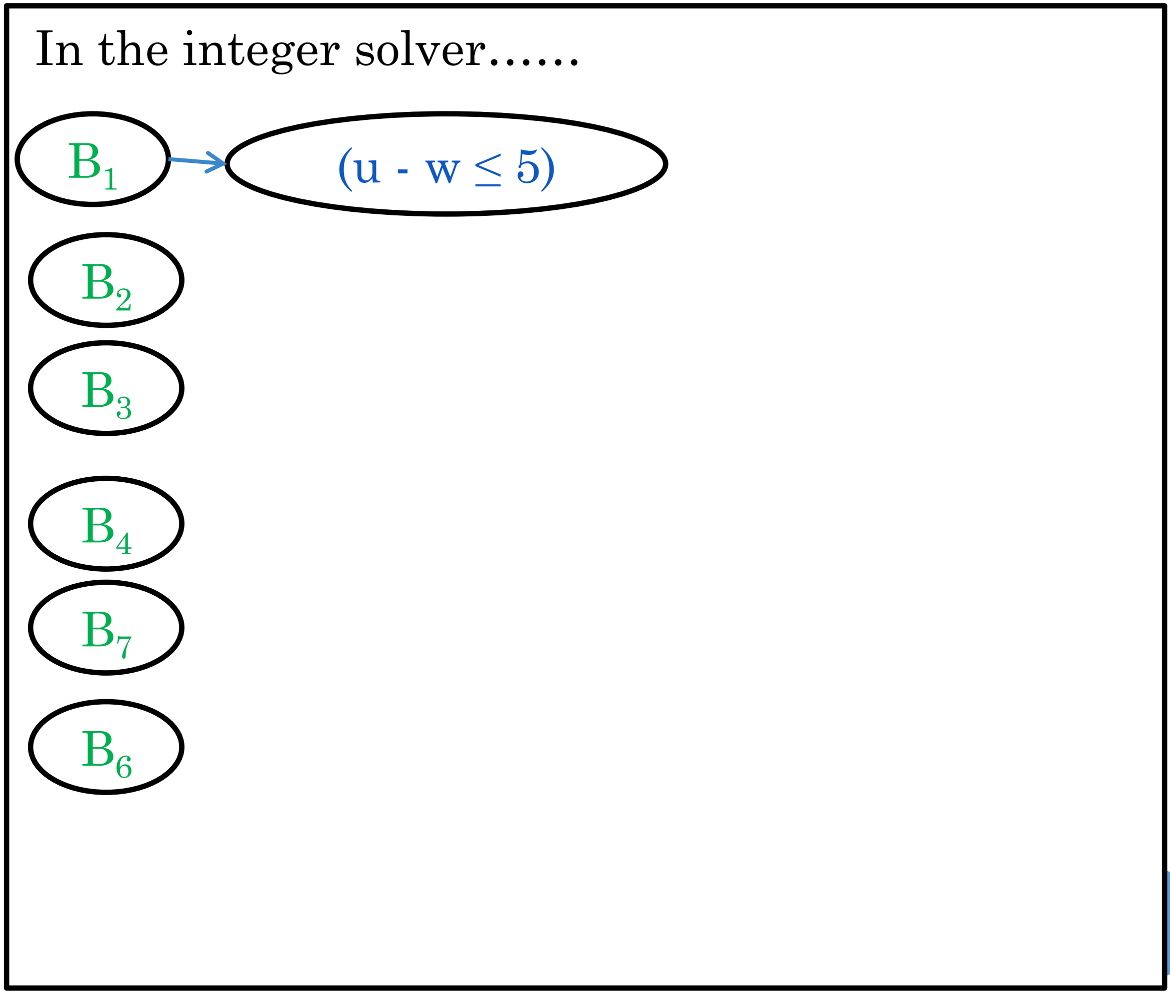
B_4

B_7

B_6

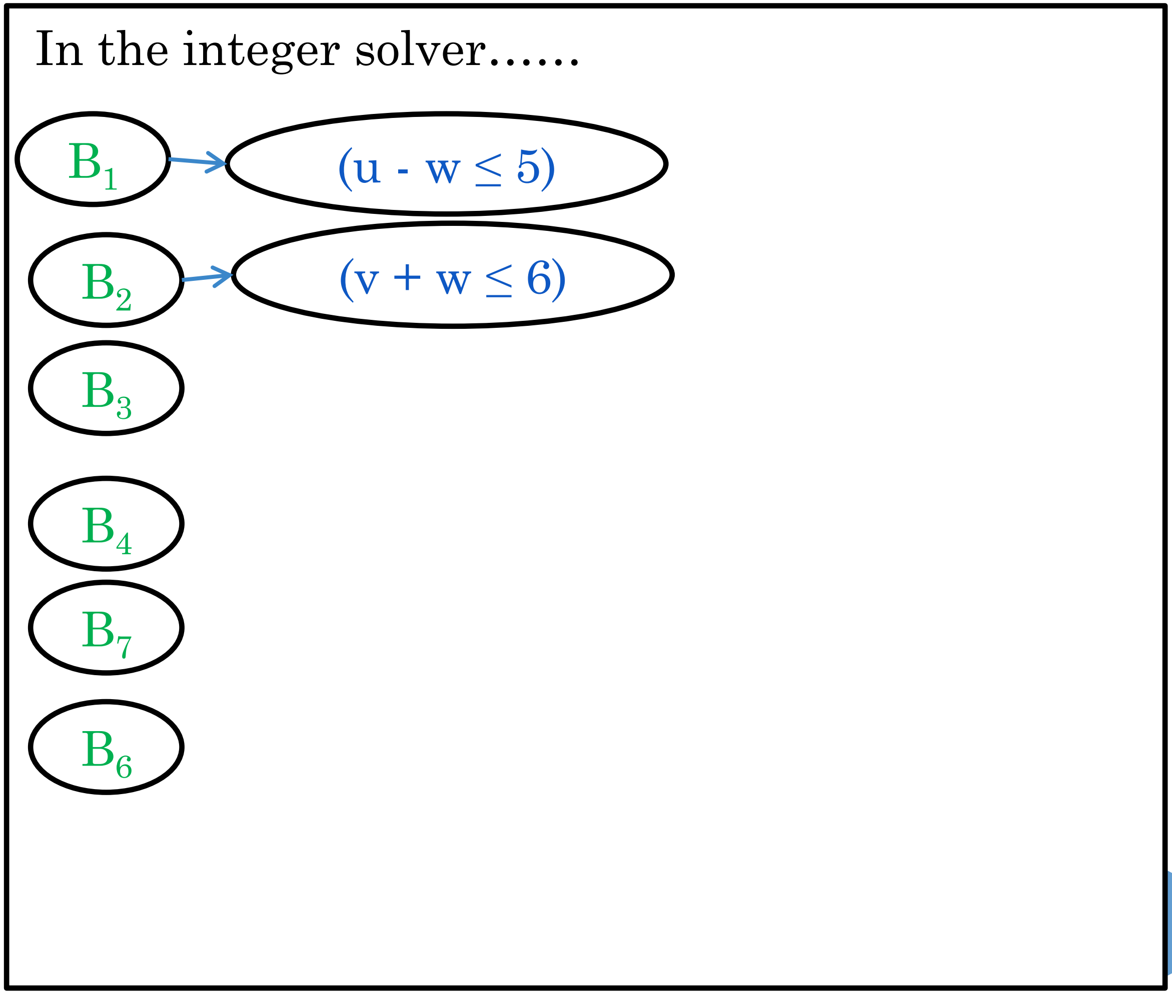
Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$



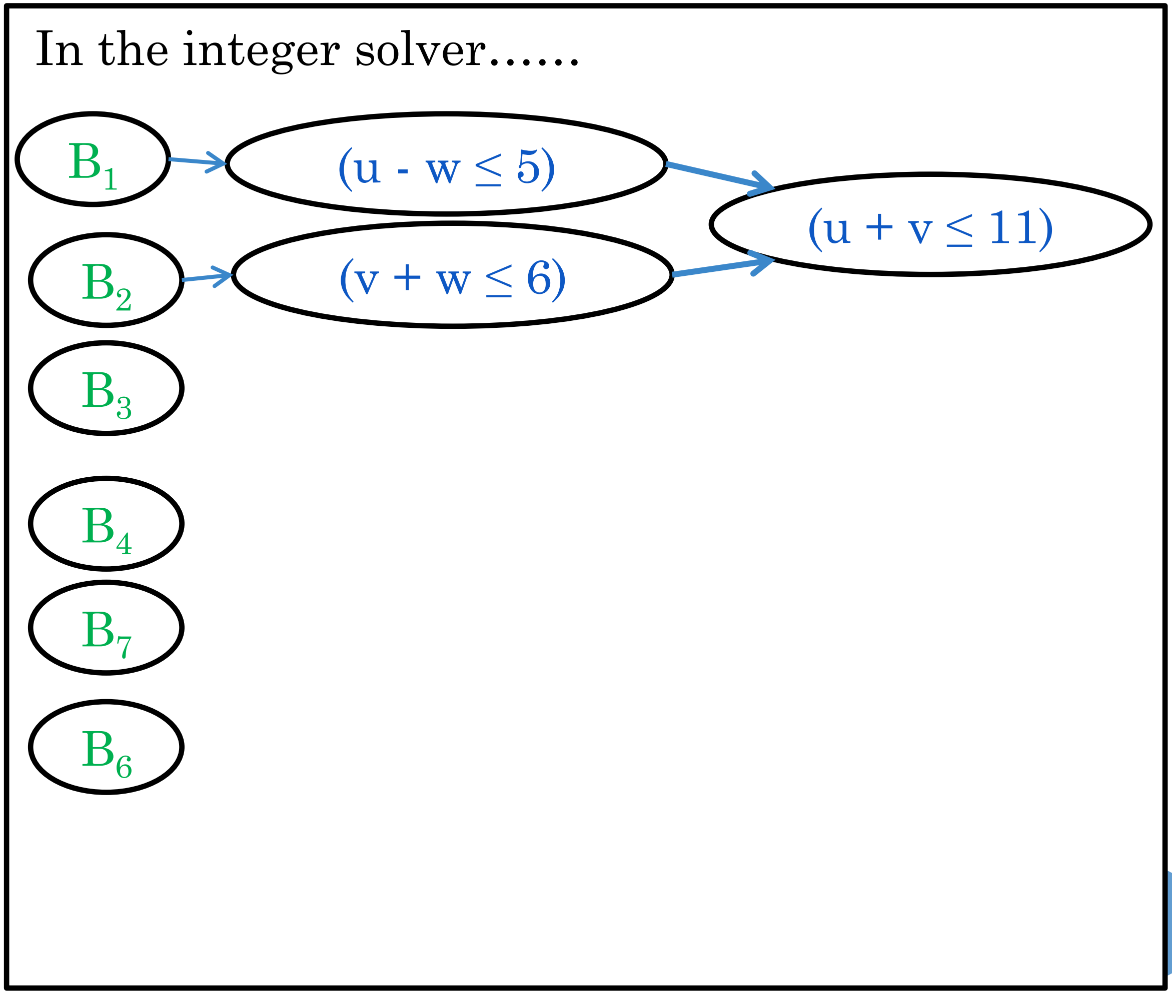
Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$



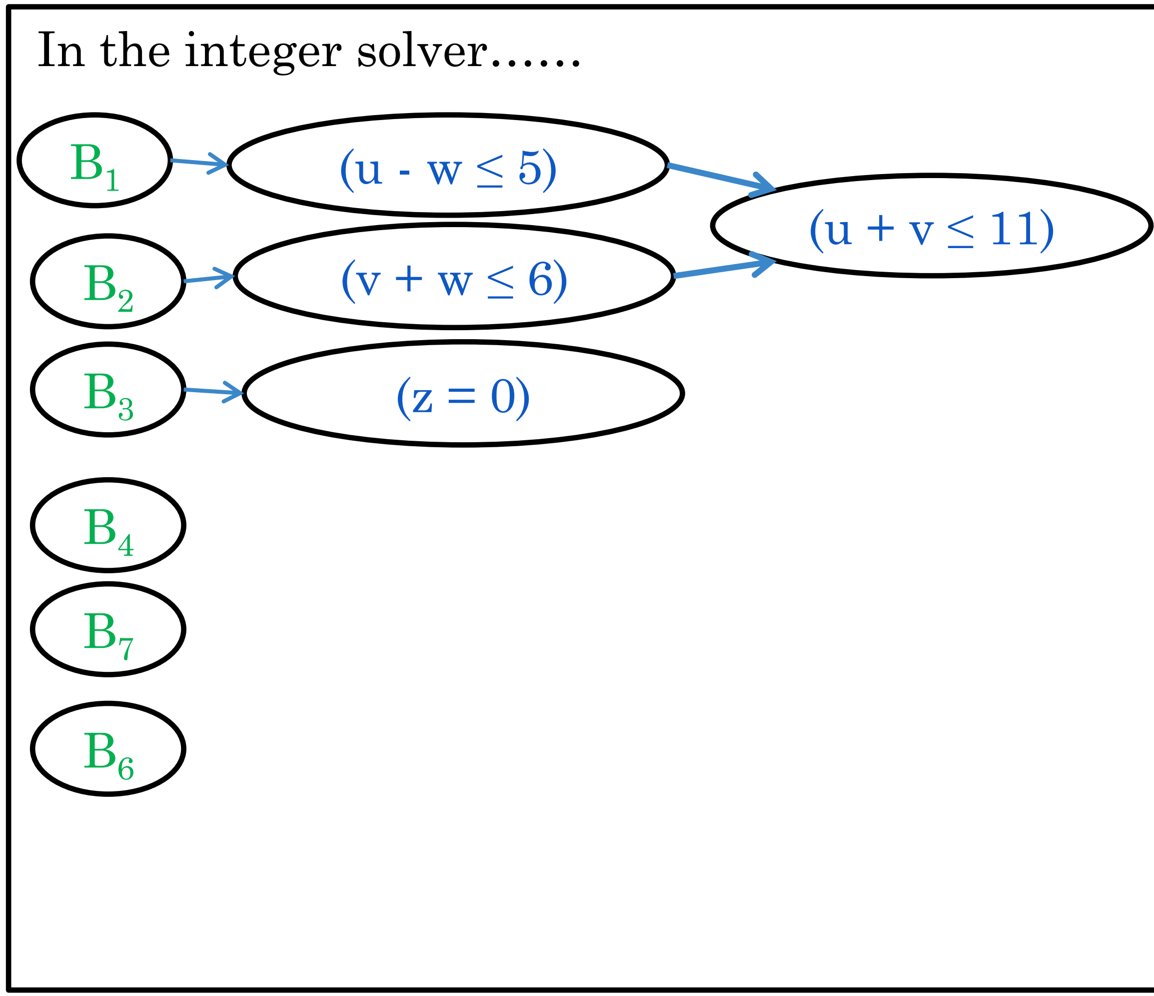
Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$



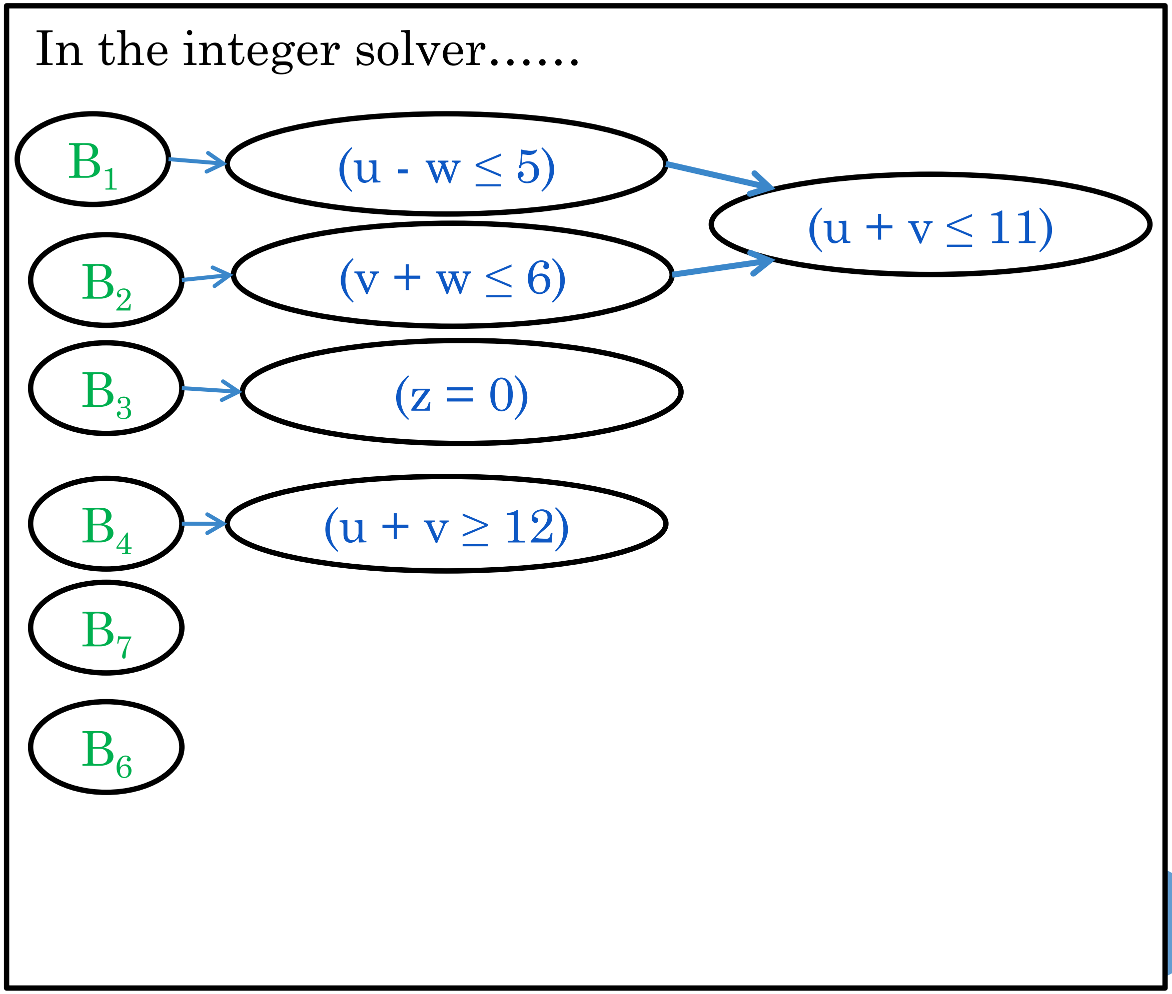
Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$



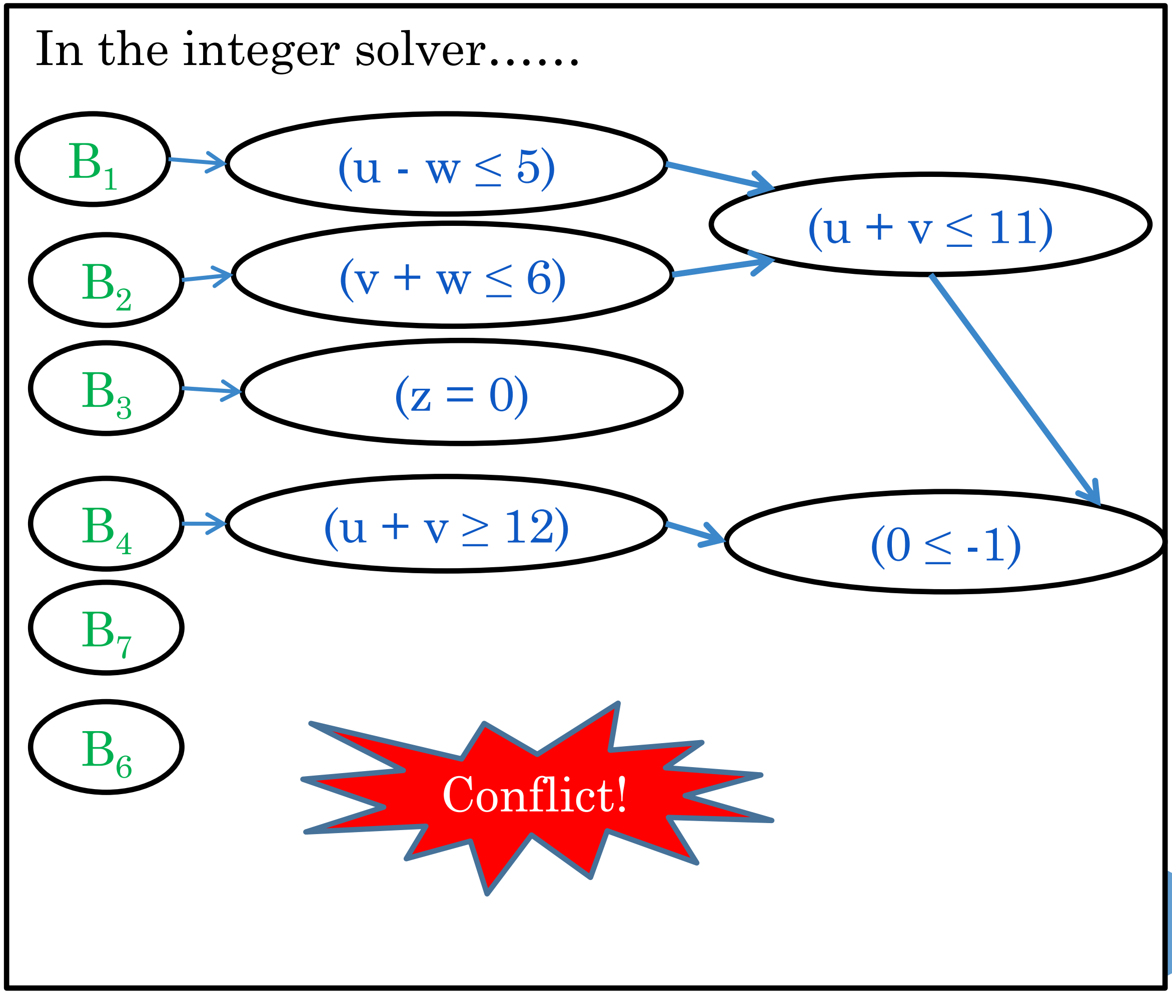
Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$



Lazy Integration(2)

$B_1 \rightarrow (u - w \leq 5)$
$B_2 \rightarrow (v + w \leq 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \geq 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$

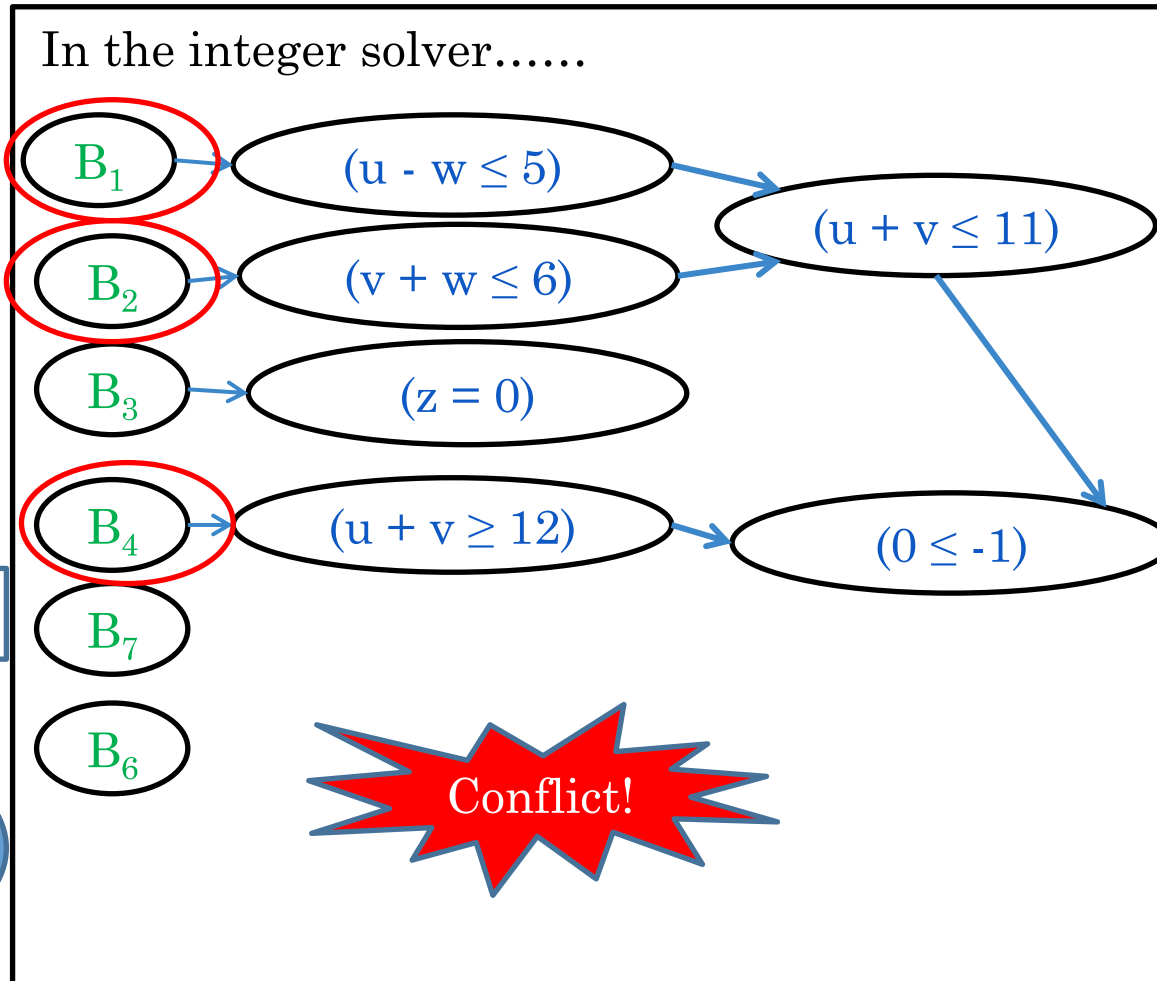


Lazy Integration(2)

$B_1 \rightarrow (u - w \le 5)$
$B_2 \rightarrow (v + w \le 6)$
$B_3 \rightarrow (z = 0)$
$B_4 \rightarrow (u + v \ge 12)$
$B_{61} \rightarrow (x = z + 1)$
$B_{71} \rightarrow (y = z + 2)$

$\neg B_1 \vee \neg B_2 \vee \neg B_4$

Add learning clause



Eager Integration

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

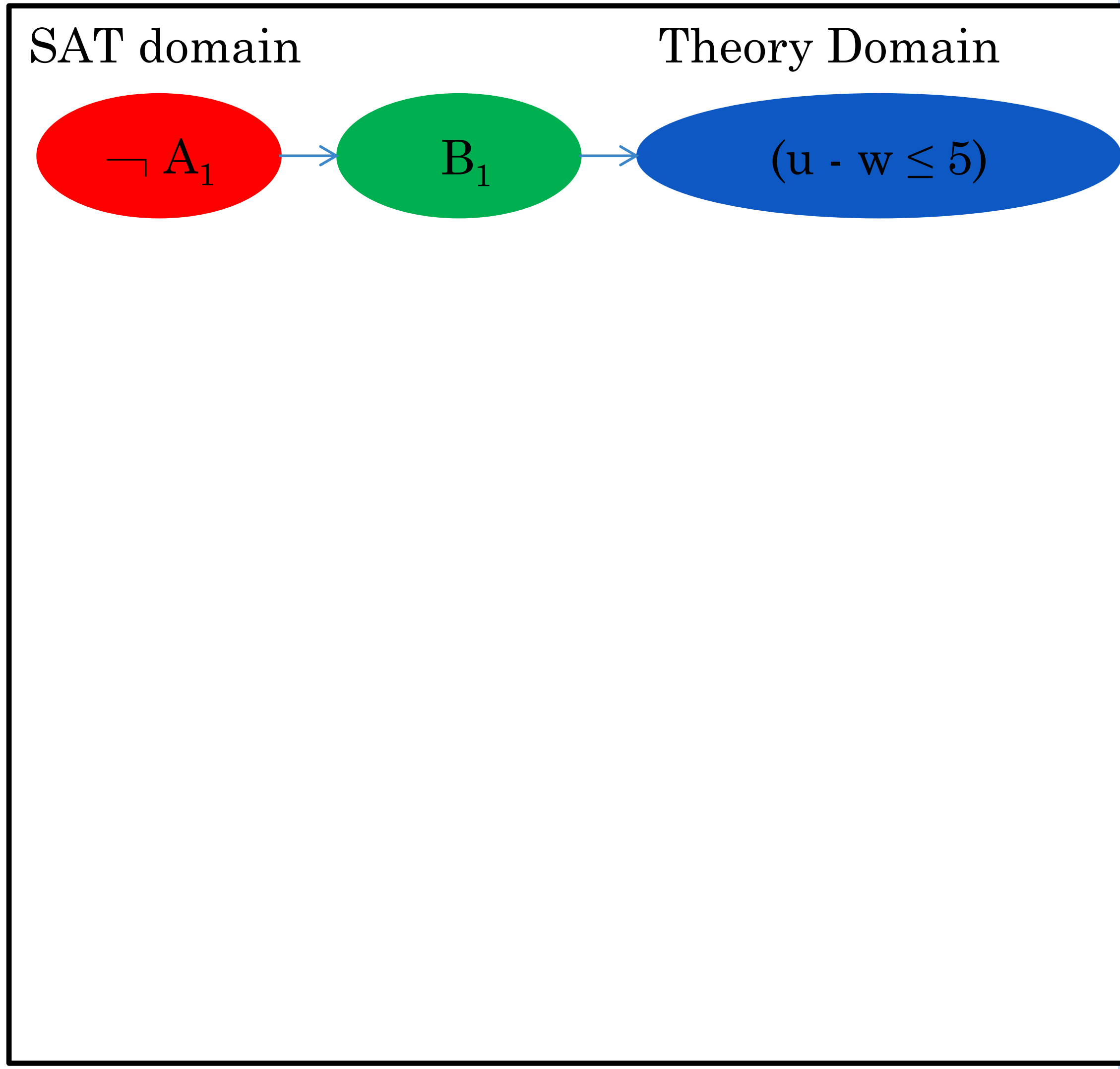
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

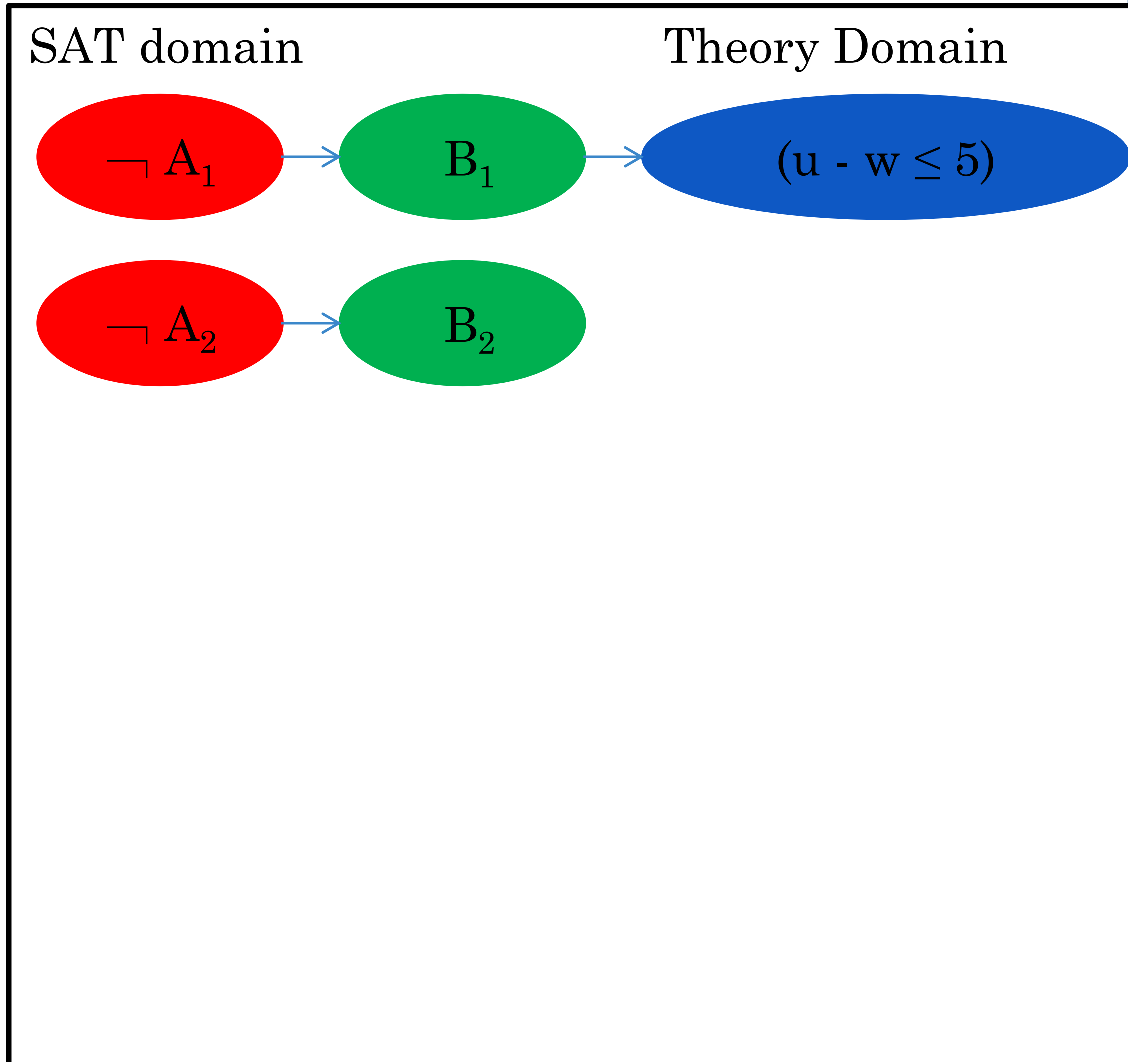
$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration

SAT instance:

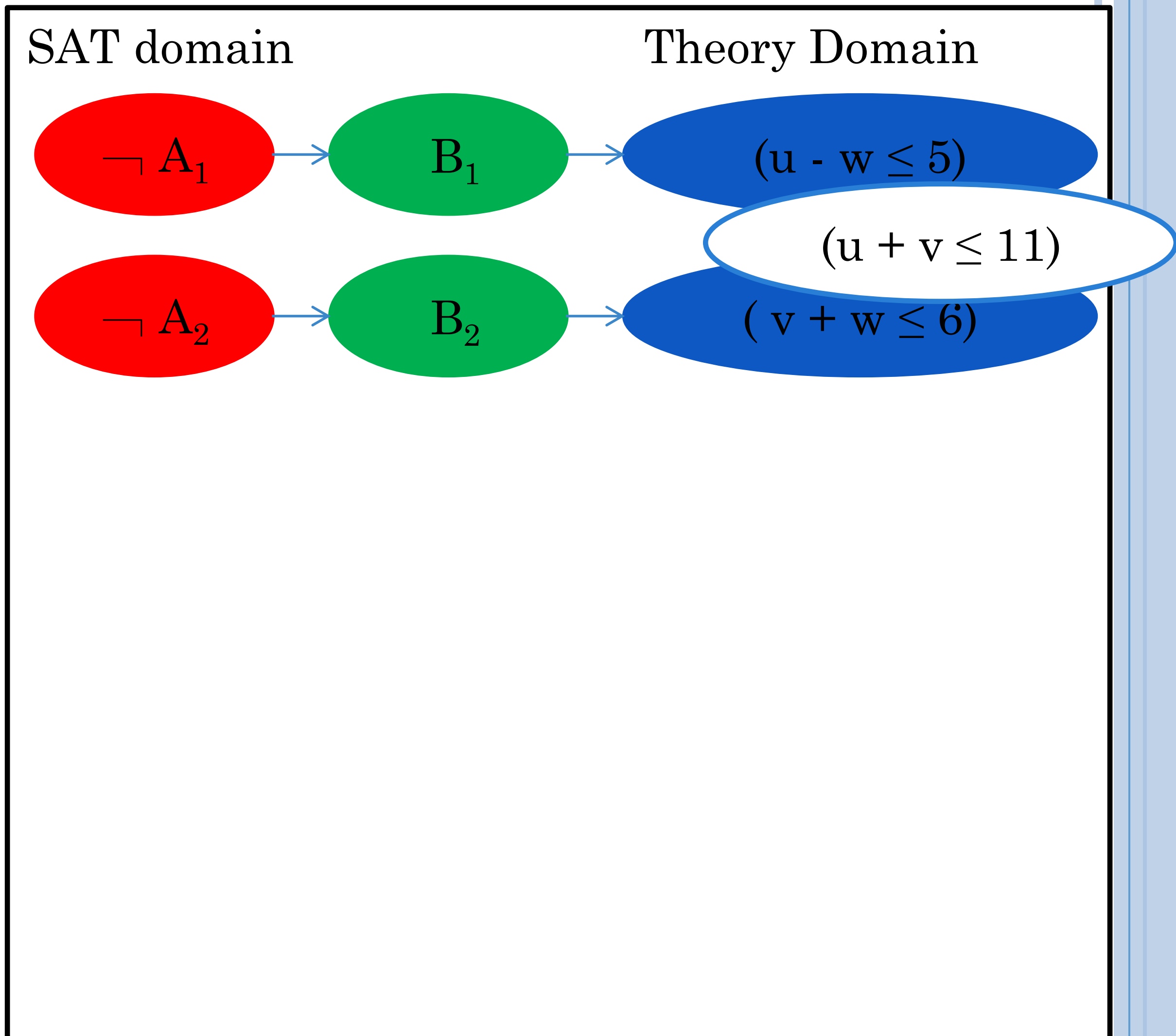
$[A_1 \vee B_1]$
$[A_2 \vee B_2]$
$[A_3 \vee B_3]$
$[A_4 \vee B_4]$
$[\neg A_3 \vee \neg A_4]$
$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$
$[B_{71} \vee B_{72} \vee B_{73}]$
$[B_1 \rightarrow (u - w \leq 5)]$
$[B_2 \rightarrow (v + w \leq 6)]$
$[B_3 \rightarrow (z = 0)]$
$[B_4 \rightarrow (u + v \geq 12)]$
$[B_{61} \rightarrow (x = z + 1)]$
$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration

SAT instance:

$[A_1 \vee B_1]$
$[A_2 \vee B_2]$
$[A_3 \vee B_3]$
$[A_4 \vee B_4]$
$[\neg A_3 \vee \neg A_4]$
$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$
$[B_{71} \vee B_{72} \vee B_{73}]$
$[B_1 \rightarrow (u - w \leq 5)]$
$[B_2 \rightarrow (v + w \leq 6)]$
$[B_3 \rightarrow (z = 0)]$
$[B_4 \rightarrow (u + v \geq 12)]$
$[B_{61} \rightarrow (x = z + 1)]$
$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

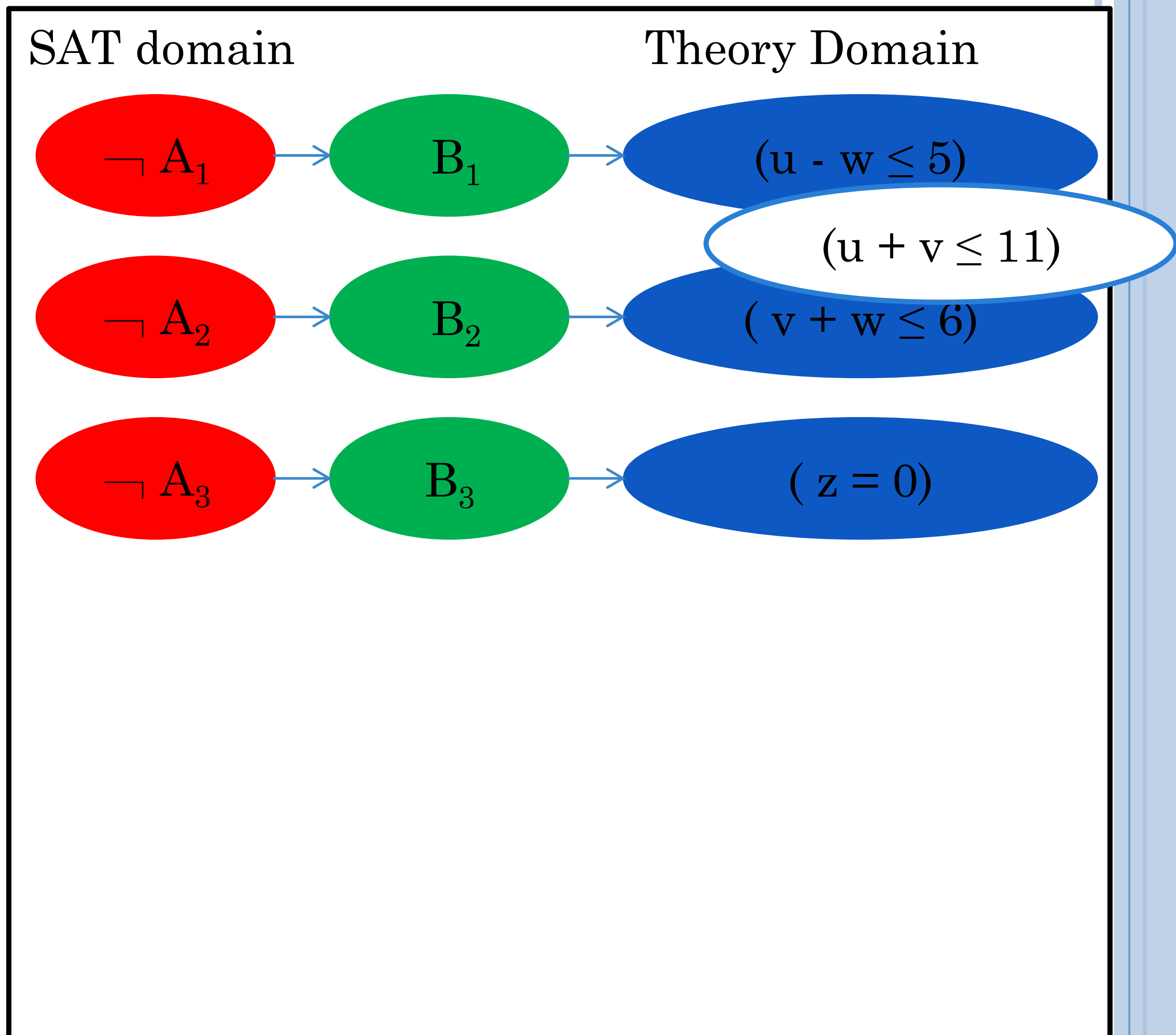
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

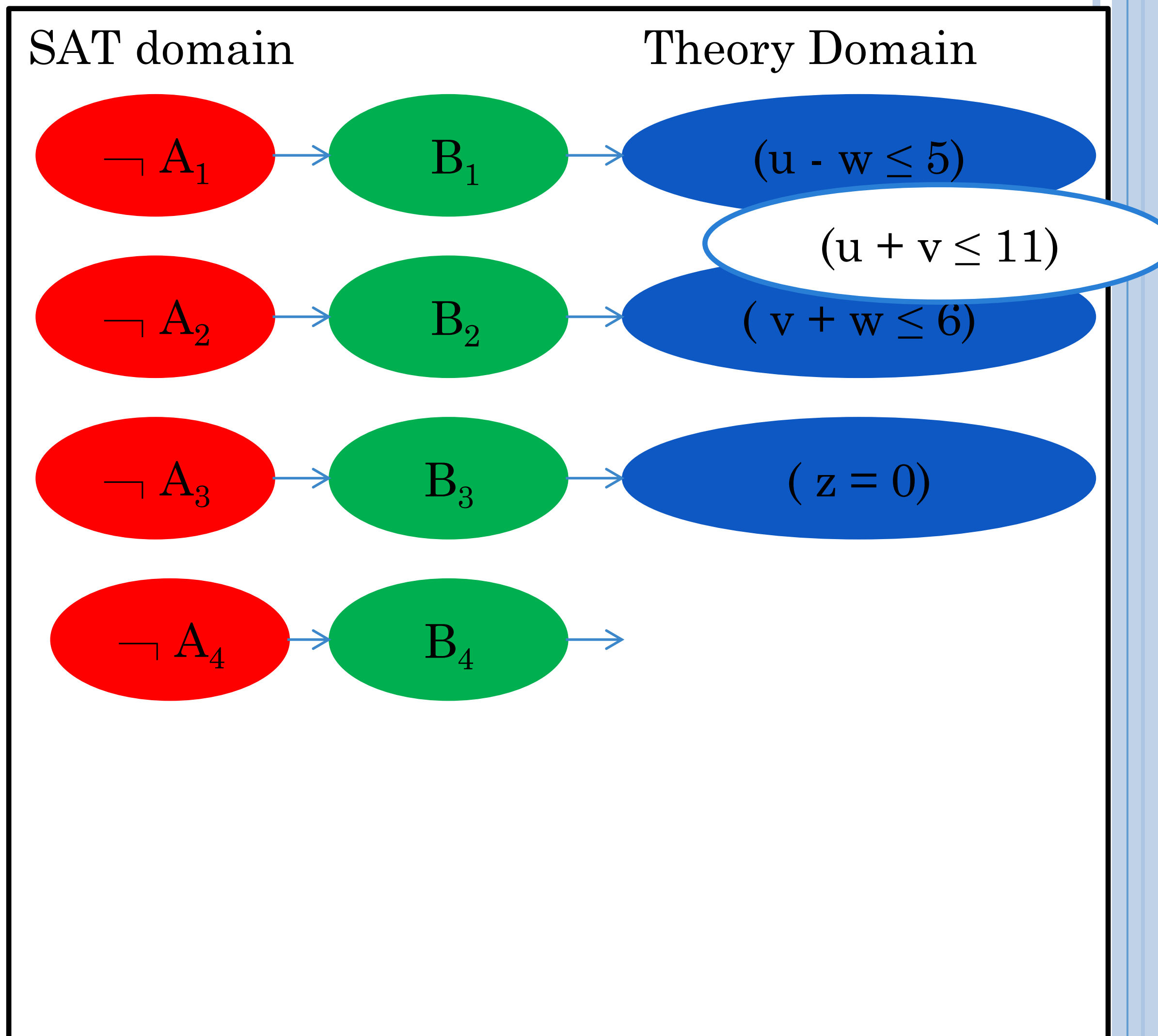
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

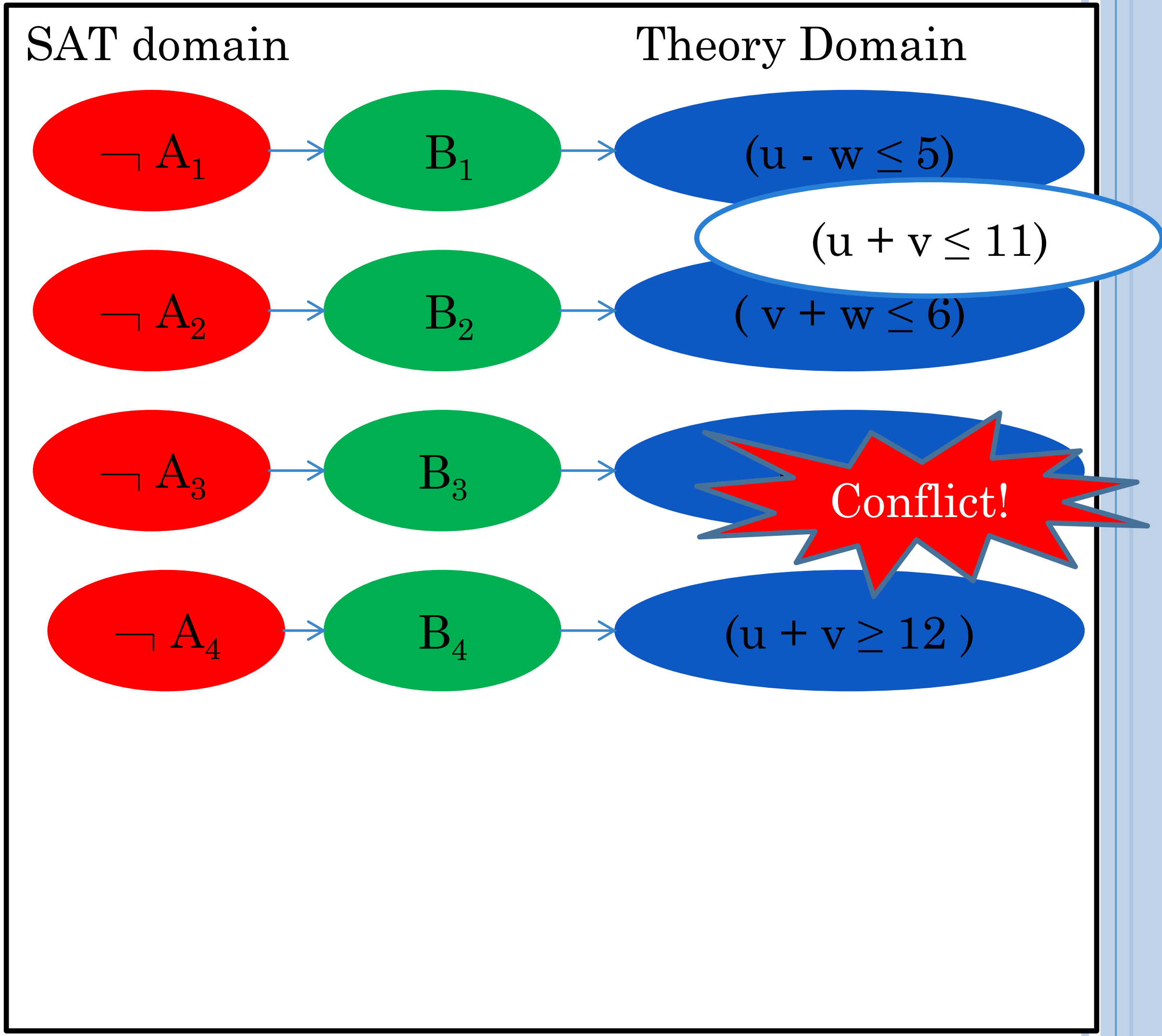
$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration

SAT instance:

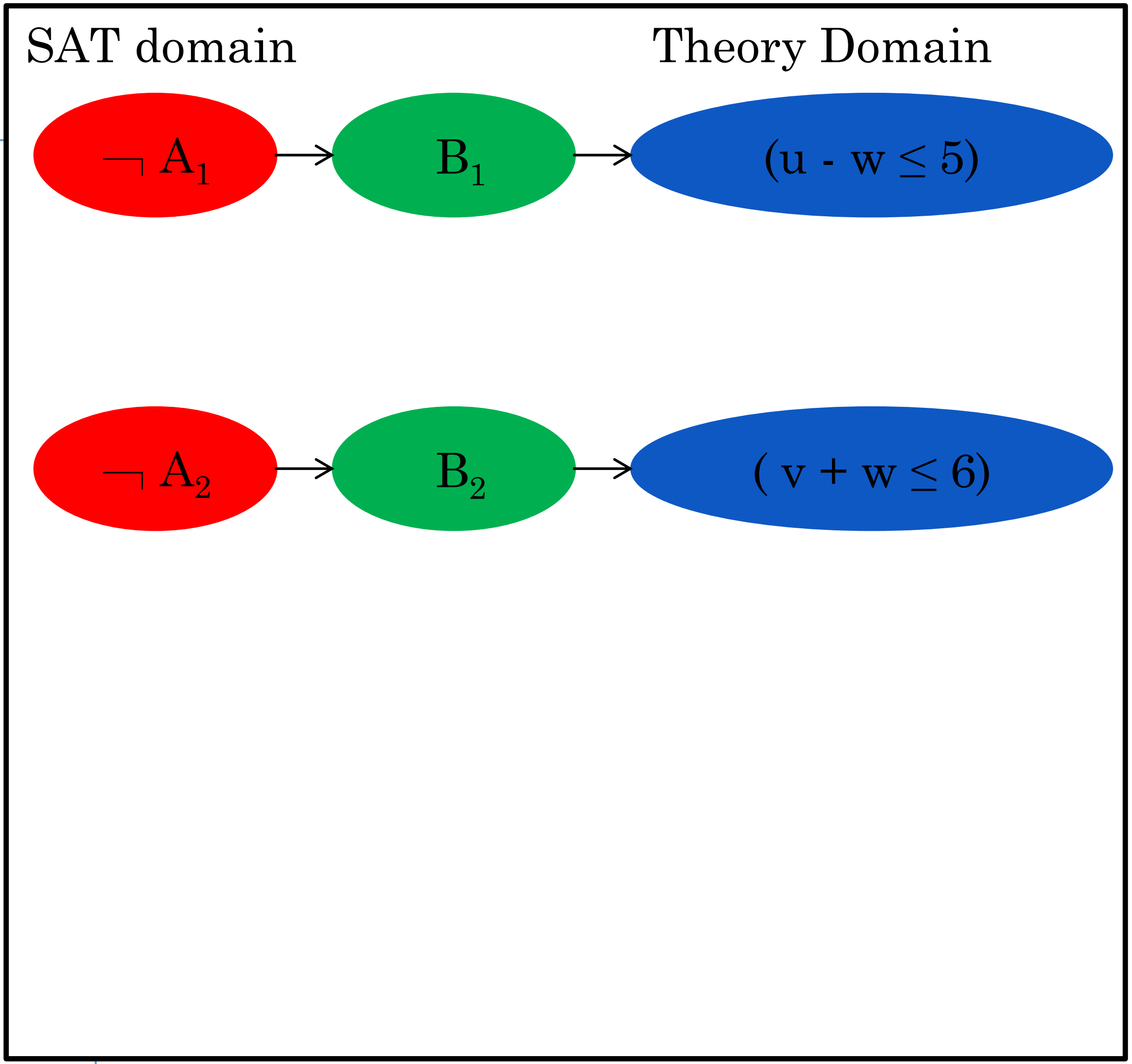
$[A_1 \vee B_1]$
$[A_2 \vee B_2]$
$[A_3 \vee B_3]$
$[A_4 \vee B_4]$
$[\neg A_3 \vee \neg A_4]$
$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$
$[B_{71} \vee B_{72} \vee B_{73}]$
$[B_1 \rightarrow (u - w \leq 5)]$
$[B_2 \rightarrow (v + w \leq 6)]$
$[B_3 \rightarrow (z = 0)]$
$[B_4 \rightarrow (u + v \geq 12)]$
$[B_{61} \rightarrow (x = z + 1)]$
$[B_{71} \rightarrow (y = z + 2)]$



EAGER INTEGRATION(2)

SAT instance:

$[A_1 \vee B_1]$
$[A_2 \vee B_2]$
$[A_3 \vee B_3]$
$[A_4 \vee B_4]$
$[\neg A_3 \vee \neg A_4]$
$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$
$[B_{71} \vee B_{72} \vee B_{73}]$
$[B_1 \rightarrow (u - w \leq 5)]$
$[B_2 \rightarrow (v + w \leq 6)]$
$[B_3 \rightarrow (z = 0)]$
$[B_4 \rightarrow (u + v \geq 12)]$
$[B_{61} \rightarrow (x = z + 1)]$
$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration(2)

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

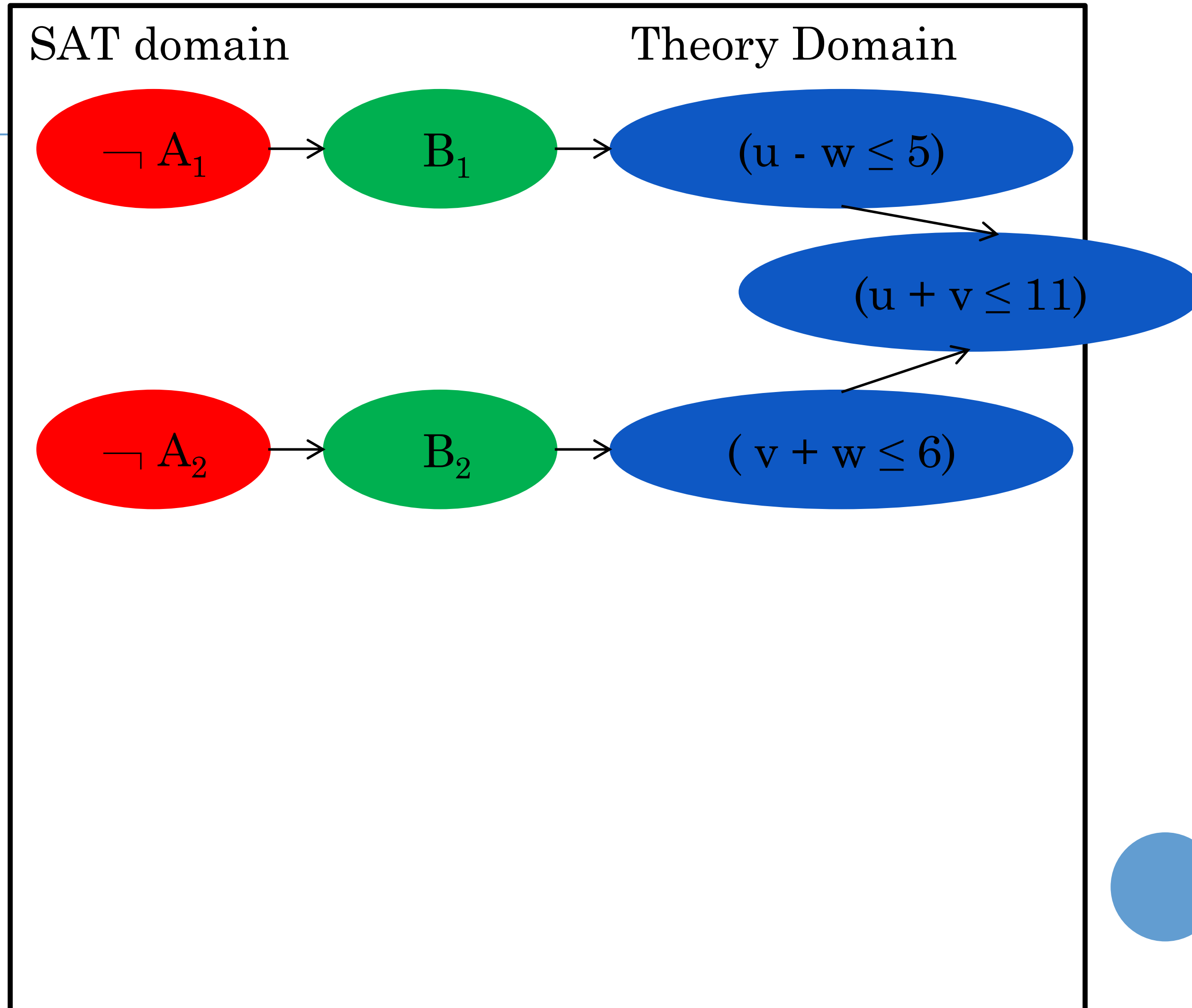
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

$[B_{71} \rightarrow (y = z + 2)]$



EAGER INTEGRATION(2)

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

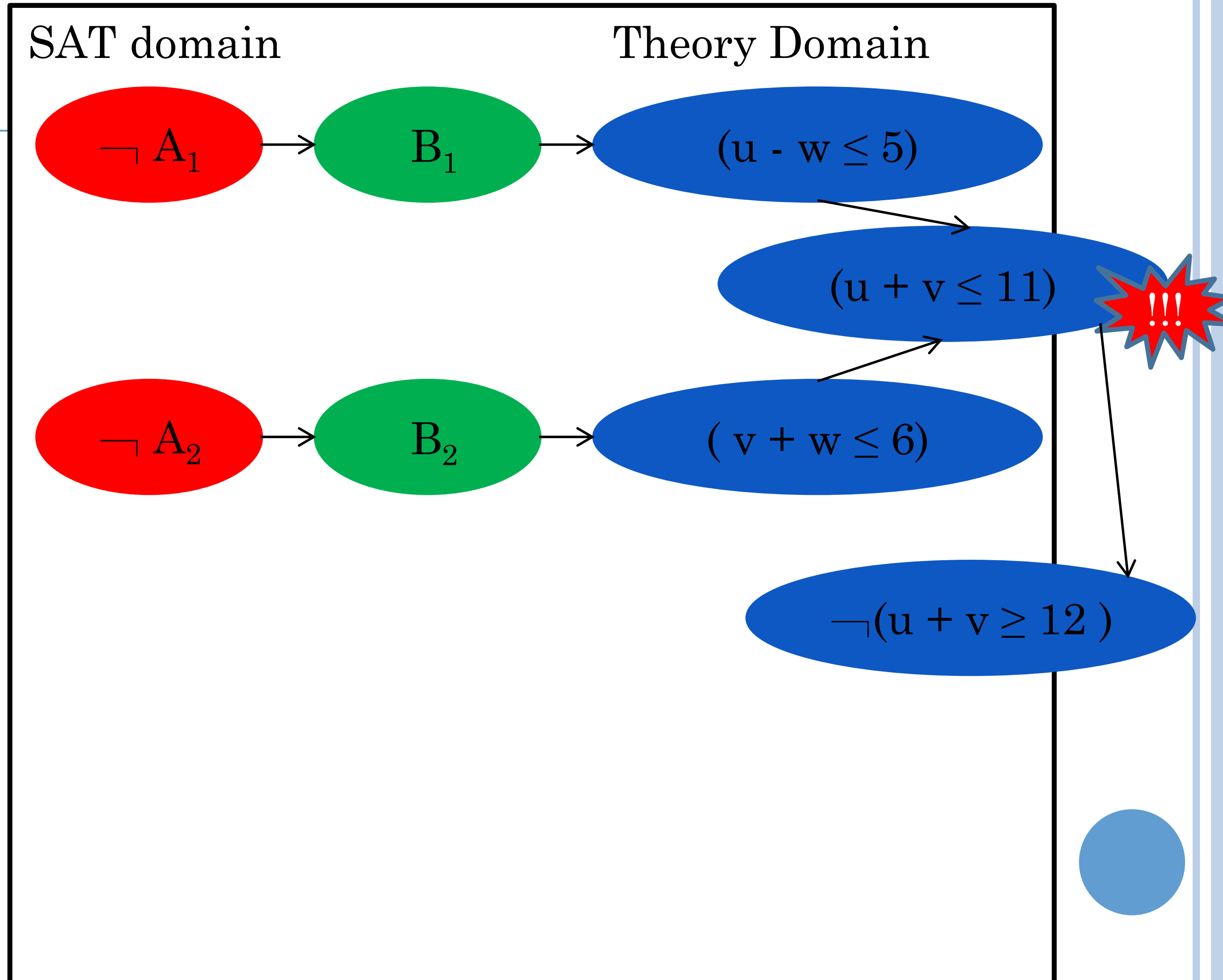
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

$[B_{71} \rightarrow (y = z + 2)]$



Eager Integration(2)

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

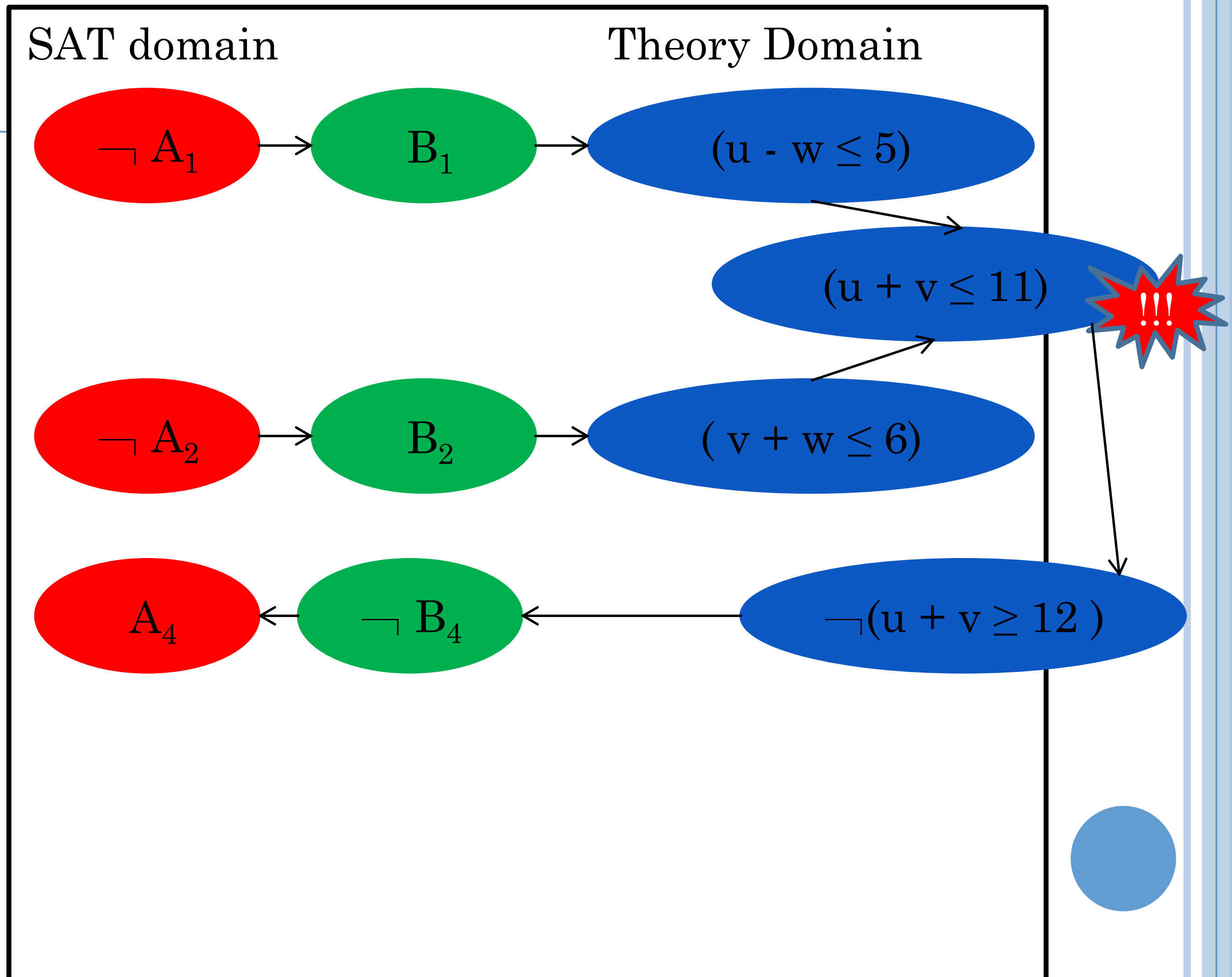
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

$[B_{71} \rightarrow (y = z + 2)]$



EAGER INTEGRATION(2)

SAT instance:

$[A_1 \vee B_1]$

$[A_2 \vee B_2]$

$[A_3 \vee B_3]$

$[A_4 \vee B_4]$

$[\neg A_3 \vee \neg A_4]$

$[B_{61} \vee B_{62} \vee B_{63} \vee B_{64}]$

$[B_{71} \vee B_{72} \vee B_{73}]$

$[B_1 \rightarrow (u - w \leq 5)]$

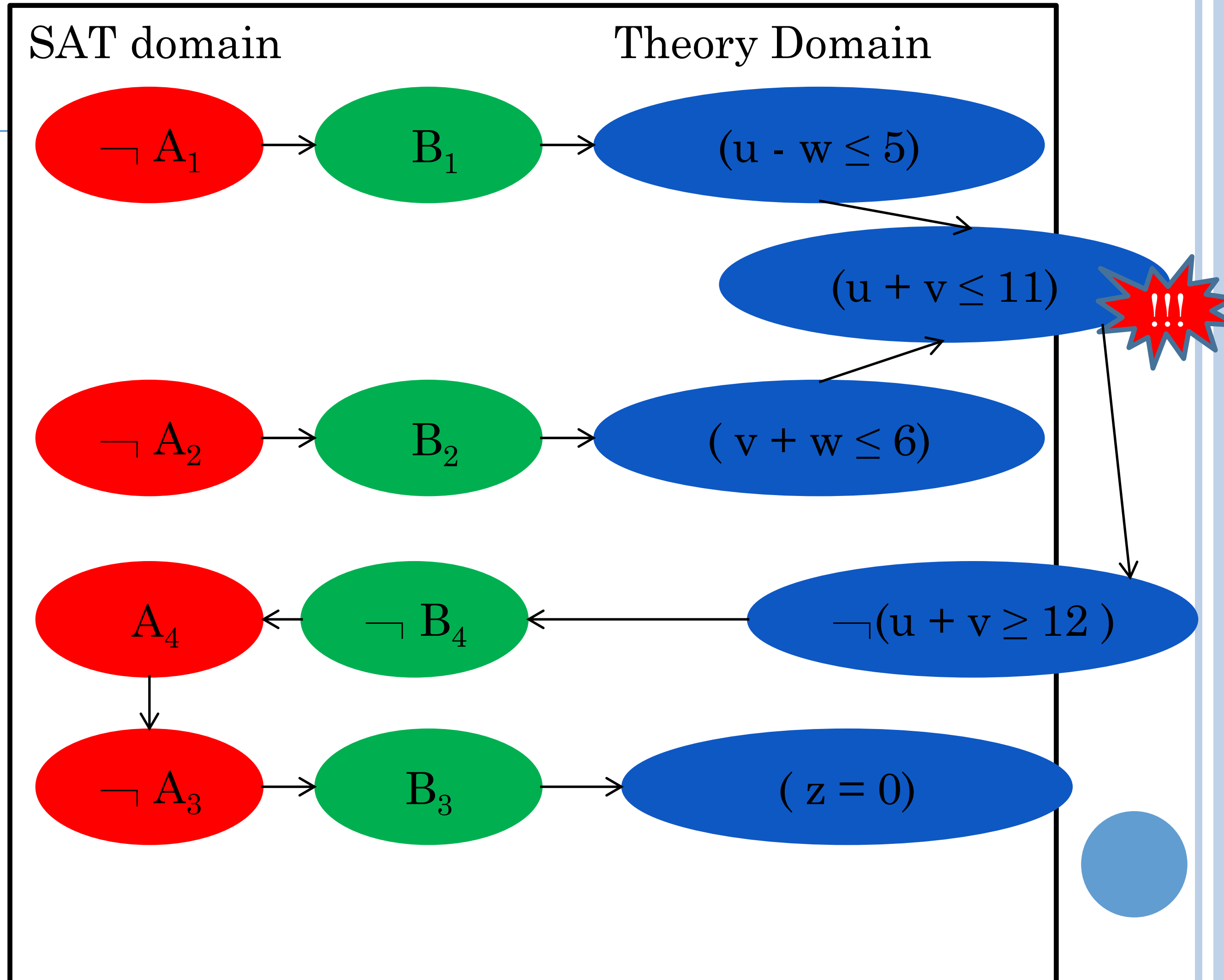
$[B_2 \rightarrow (v + w \leq 6)]$

$[B_3 \rightarrow (z = 0)]$

$[B_4 \rightarrow (u + v \geq 12)]$

$[B_{61} \rightarrow (x = z + 1)]$

$[B_{71} \rightarrow (y = z + 2)]$



Comparison

- Lazy Integration
  - SAT solver can work as an enumerator.
  - ☺Easier to implement
  - ☹Can not find out the conflicts earlier
- Eager Integration
  - Requires a tighter integration of the source codes of the SAT solver and $\mathcal{T}-$solver.
  - ☺Able to detect conflict earlier
  - ☹ terrible implementation
- The choice relies on the trade-off between efficiency and implementation effort.

Resource of MathSAT5

- Italy, University of Trento
  - http://mathsat.fbk.eu/
  - 3 Ph.D. thesis and many papers…
- Only execution file and libraries(C API)…
- Provide some API to use the
  - Read problem in smt2 format
    - (and (= v3 (h v0)) (= v4 (h v1)) (= v6 (f v2)) (= v7 (f v5)))

Specialty of MathSAT

- Layered theory solvers
- Sometimes a fully general solver for $\mathcal{T}$ is not always needed.
  - For example, difference constraints are special case of linear constraints, and are easier to be solved.
- Thus, a $\mathcal{T}$-solver may be organized in a layered hierarchy of solvers of increasing solving capabilities.
- Ex: Difference → UTVPI → Linear
  - UTVPI: two integer variables per inequality constraint
    - $a*x+b*y < c$

Resource of Z3

- Create by MicroSoft
  - http://research.microsoft.com/en-us/um/redmond/projects/z3/
- Only execution file and libraries…
- Has been used in several program analysis, verification, test case generation projects at Microsoft
- Support Several input formats
  - SMT-LIB, Z3, Dimacs
- Main features
  - Linear real and integer arithmetic.
  - Fixed-size bit-vectors
  - Uninterpreted functions
  - Extensional arrays
  - Quantifiers

Combination of theories

- Example:
  - $x+2 = y \Rightarrow f(read(write(a, x, 3), y - 2)) = f(y - x + 1)$
- Given
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $\mathcal{T}_1, \mathcal{T}_2$: theories over Σ_1, Σ_2
  - $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$
- Is $\mathcal{T}$ consistent?
- Given satisfiability procedures for conjunction of literals of $\mathcal{T}1$ and $\mathcal{T}2$, how to decide the satisfiability of $\mathcal{T}$?
- Nelson-Oppen Combination

Combination of theories(2)

- Nelson-Oppen Combination
- Essential concept:
  - Purification
    - For a conjunction of $(\Sigma_1 \cup \Sigma_2)$-literals φ, transform it into a equisatisfiable $\phi_1 \wedge \phi_2$ such that ϕ_i contains only Σ_i-literals.
  - Stably-Infinite Theories
    - A theory is stably infinite if every satisfiable sentence is satisfiable in an infinite model.
    - Example: Theories with only finite models are not stably infinite. (only two elements in the domain)
      - T = (∀x, y, z. (x = y) ∨ (x = z) ∨ (y = z)).
    - The union of two consistent, disjoint, stably infinite theories is consistent.
  - Convex Theories.
    - for all finite sets Γ of literals and for all non-empty disjunctions $\bigvee_{i \in I} x_i = y_i$ of variables
    - $\Gamma \vDash_T \bigvee_{i \in I} x_i = y_i$ iff $\Gamma \vDash_T x_i = yi$for some i $\in I$

Nelson-Oppen Combination

- Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be consistent, stably infinite theories over disjoint (countable) signatures.
- Assume satisfiability of conjunction of literals can decided in $O(T_1(n))$ and $O(T_2(n))$ time respectively. Then,
  - 1. The combined theory T is consistent and stably infinite.
  - 2. Satisfiability of quantifier free conjunction of literals in $\mathcal{T}$ can be decided in $O(2^{n^2} \times (T_1(n) + T_2(n))$.
  - 3. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex, then so is $\mathcal{T}$ and satisfiability in $\mathcal{T}$ is in $O(n^3 \times (T_1(n) + T_2(n)))$.

Nelson-Oppen Combination Procedure

- Initial State:
  - φ is a conjunction of literals over $\Sigma_1 \cup \Sigma_2$.
- Purification:
  - Preserving satisfiability transform φ into $\varphi_1 \wedge \varphi_2$, such that, $\varphi_i \in \Sigma_i$
- Interaction:
  - Guess a partition of $V(\varphi 1) \cap V(\varphi 2)$ into disjoint subsets. Express it as conjunction of literals ψ.
  - Example. The partition $\{x_1\}, \{x_2, x_3\}, \{x_4\}$ is represented as
    - $x_1 \neq x_2, x_1 \neq x_4, x_2 \neq x_4, x_2 = x_3$.
- Component Procedures :
  - Use individual procedures to decide whether $\varphi i \wedge \psi$ is satisfiable
- Return:
  - If both return yes, return yes. No, otherwise.

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$$(x + 2 = y) \wedge f(read(write(a, x, 3), y - 2)) \neq f(y - x + 1)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{LA}$	$\mathcal{T}_{\mathrm{AR}}$

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$(x + 2 = y) \wedge f(read(write(a, x, 3), y - 2)) \neq f(y - x + 1)$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$$f(read(write(a, x, 3), y - 2)) \neq f(y - x + 1)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
	$x + 2 = y$	

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$$f(read(write(a, x, u_1), y - 2)) \neq f(y - x + 1)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\text{LA}}$	$\mathcal{T}_{\text{AR}}$
	$x + 2 = y$ $u_1 = 3$	

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$$f(\,read(write(a, x, u_1), u_2)\,) \neq f(y - x + 1)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$	

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$$f(u_3) \neq f(y - x + 1)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{LA}$	$\mathcal{T}_{\mathrm{AR}}$
	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$	$u_3 = read(write(a, x, u_1), u_2)$

NO PROCEDURE: EXAMPLE

Purifying

Problem :

$$f(u_3) \neq f(u_4)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
	$x + 2 = y$	$u_3 = read(write(a, x, u_1), u_2)$
	$u_1 = 3$	
	$u_2 = y - 2$	
	$u_4 = y - x + 1$	

NO PROCEDURE: EXAMPLE

Finish Purifying

Problem :

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$	$u_3 = read(write(a, x, u_1), u_2)$

NO PROCEDURE: EXAMPLE

Solving $\mathcal{T}_{\mathrm{LA}}$

Problem :

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = read(write(a, x, u_1), u_2)$
	$u_1 = 3$	
	$u_2 = y - 2$	
	$u_4 = y - x + 1$	

NO PROCEDURE: EXAMPLE

Propagate
$u_2 = x$

Problem :

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = read(write(a, x, u_1), u_2)$
	$u_1 = 3$	
	$u_2 = x$	
	$u_4 = 3$	

NO PROCEDURE: EXAMPLE

Solve $\mathcal{T}_{\mathrm{AR}}$

Problem :

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = read(write(a, x, u_1), u_2)$
	$u_1 = 3$	
$u_2 = x$	$u_2 = x$	$u_2 = x$
	$u_4 = 3$	

NO PROCEDURE: EXAMPLE

Propagate
$u_3 = u_1$

Problem :

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = u_1$
	$u_1 = 3$	
$u_2 = x$	$u_2 = x$	$u_2 = x$
	$u_4 = 3$	

NO PROCEDURE: EXAMPLE

Propagate
$u_4 = u_1$

Problem :

$$u_1 = 3 \wedge u_4 = 3 \Rightarrow u_4 = u_1$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{LA}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
$u_3 = u_1$	$u_2 = x$	
	$u_4 = 3$	
	$u_3 = u_1$	

NO PROCEDURE: EXAMPLE

Solve

Problem :

$$\text{Congruence } u_3 = u_1 \wedge u_4 = u_1 \Rightarrow f(u_3) = f(u_4)$$

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{A}$	$\mathcal{T}_{AR}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
$u_3 = u_1$	$u_2 = x$	$u_4 = u_1$
$u_4 = u_1$	$u_4 = 3$	
$f(u_3) = f(u_4)$	$u_3 = u_1$	
	$u_4 = u_1$	

NO PROCEDURE: EXAMPLE

Solve

Problem :

UNSAT!

$\mathcal{T}_{\mathbb{E}}$	$\mathcal{T}_{\mathrm{A}}$	$\mathcal{T}_{\mathrm{AR}}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
$u_3 = u_1$	$u_2 = x$	$u_4 = u_1$
$u_4 = u_1$	$u_4 = 3$	
$f(u_3) = f(u_4)$	$u_3 = u_1$	
	$u_4 = u_1$	

Conclusion

- We go through
  - some theories of interest
  - eager approaches to SMT
  - lazy approaches to SMT
- Some theories and algorithms are simply discussed
- More details: see reference slides.

Reference

- **A Mathematical Introduction to Logic**
  - **by Herbert B. Enderton**
- SMT-COMP
  - The Satisfiability Modulo Theories Competition
  - http://smtcomp.sourceforge.net/2012/
- SMT-LIB
  - The Satisfiability Modulo Theories Library
  - http://goedel.cs.uiowa.edu/smtlib/
- Satisfiability Modulo Theories slides
  - Roberto Sebastiani for IJCAI 11
- Solvers' websites:
  - Boolector, Yices, Z3, MathSAT5
  - Many papers from MathSAT team
  - Tutorial slides from Z3
- Previous slides from Yi-Wen Chang and Chih-Chun Lee
- Congruence closure
  - http://www.cs.berkeley.edu/~necula/autded/lecture12-congclos.pdf
- Difference Logic
  - http://www.lsi.upc.edu/~oliveras/TDV/dl.pdf