DPLL(T):Fast Decision Procedures

Harald Ganzinger	George Hagen	Robert Nieuwenhuis
	Cesare Tinelli	Albert Oliveras
MPI, Saarburcken	The University of Iowa	UPC, Barcelona

Computer Aided-Verification (CAV)

Boston, July 2004

In Memoriam

Harald Ganzinger (1950-2004)

Overview of this talk

1. Introduction
2. Lazy vs eager approach
   - Lazy approach: advantages and disadvantages
   - Eager approach: advantages and disadvantages
3. DPLL(T): Our framework for SMT
   - The DPLL algorithm
   - Branching heuristics, unit propagation and conflict analysis
   - Comparison with existing approaches
4. A concrete case: EUF with offsets
   - A solver for EUF
   - Experimental results
5. Conclusions and future work

SMT: Satisfiability modulo theories

$$g(a)=c \;\wedge\; (\, f(g(a))\neq f(c) \,\vee\, g(a)=d \,) \;\wedge\; c\neq d$$

- Theories of interest: EUF [Burch and Dill '94], CLU [Bryant, Lahiri and Seshia '02], separation logic [BLS '03], arrays, ...
- Applications: circuit design, compiler optimization, planning, scheduling, software/hardware verification, ...

Lazy vs eager approach

Lazy approach

The following three steps are iterated

- SAT solver looks for a propositional model
- Specialized procedure for conjunctions of literals checks its consistency
- If model consistent then formula is SAT, otherwise a lemma is added precluding the model

– constraints imposed by the theory introduced on demand

– Lazy/eager notification, online/offline SAT solver, extraction of inconsistency proofs [Armando et al ’00; deMoura and Ruess ’02; Barret, Dill and Stump ’02; Flanagan et al ’03, etc]

Lazy vs eager approach

Lazy approach

- Advantages:
  - Use of off-the-shelf theory solvers
  - Can use of almost off-the-shelf SAT solvers
- Disadvantages:
  - Information from the theory only used to validate propositional models
  - Too many iterations may be required
- Tools: SVC, CVC (Lite), ICS, VeriFun, MathSAT

Lazy vs eager approach

Eager approach

- formula converted into an equisatisfiable propositional one to be checked by a SAT solver
- Two steps (for CLU)
  - Functional symbols are removed, only constants left
  - (in)Equality is removed
- Small-domain encoding (SD) [Pnuelli et al '99, BLS '02], Per-constraint encoding (EIJ) [Bryant, German and Velev '02; Bryant and Velev '02], Hybrid methods [BLS '02, '03]

Lazy vs eager approach

Eager approach: different encodings

Given the equality formula:

$$(k_1 = k_2 \lor k_3 = k_4) \land (k_2 = k_3 \lor k_1 = k_4 \lor k_2 = k_4)$$

Small Domain encoding (SD): propositional formula small but suffers from loss of structure

$$(x_{11} \lor ((x_{31} \land x_{41}) \lor (\neg x_{31} \land x_{32} \land \neg x_{41}))) \land$$
$$(x_{31} \lor ((x_{11} \land x_{41}) \lor (\neg x_{11} \land x_{12} \land \neg x_{41})) \lor x_{41})$$

Per-constraing encoding (EIJ): structure preserved but size may be exponential if pred/succ allowed

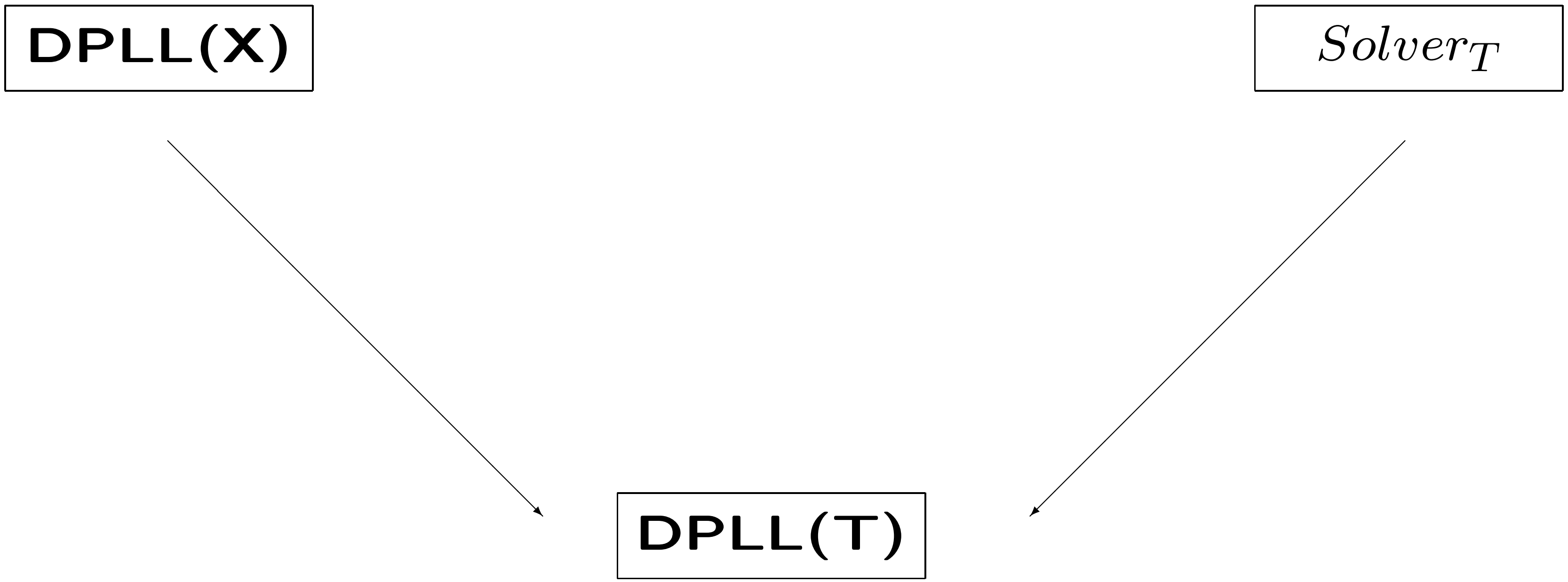
$$(e_{12} \lor e_{34}) \land (e_{23} \lor e_{14} \lor e_{24})$$
$$e_{12} \land e_{24} \Rightarrow e_{14}$$
$$e_{12} \land e_{14} \Rightarrow e_{24}$$
$$\ldots$$

Lazy vs eager approach

Eager approach

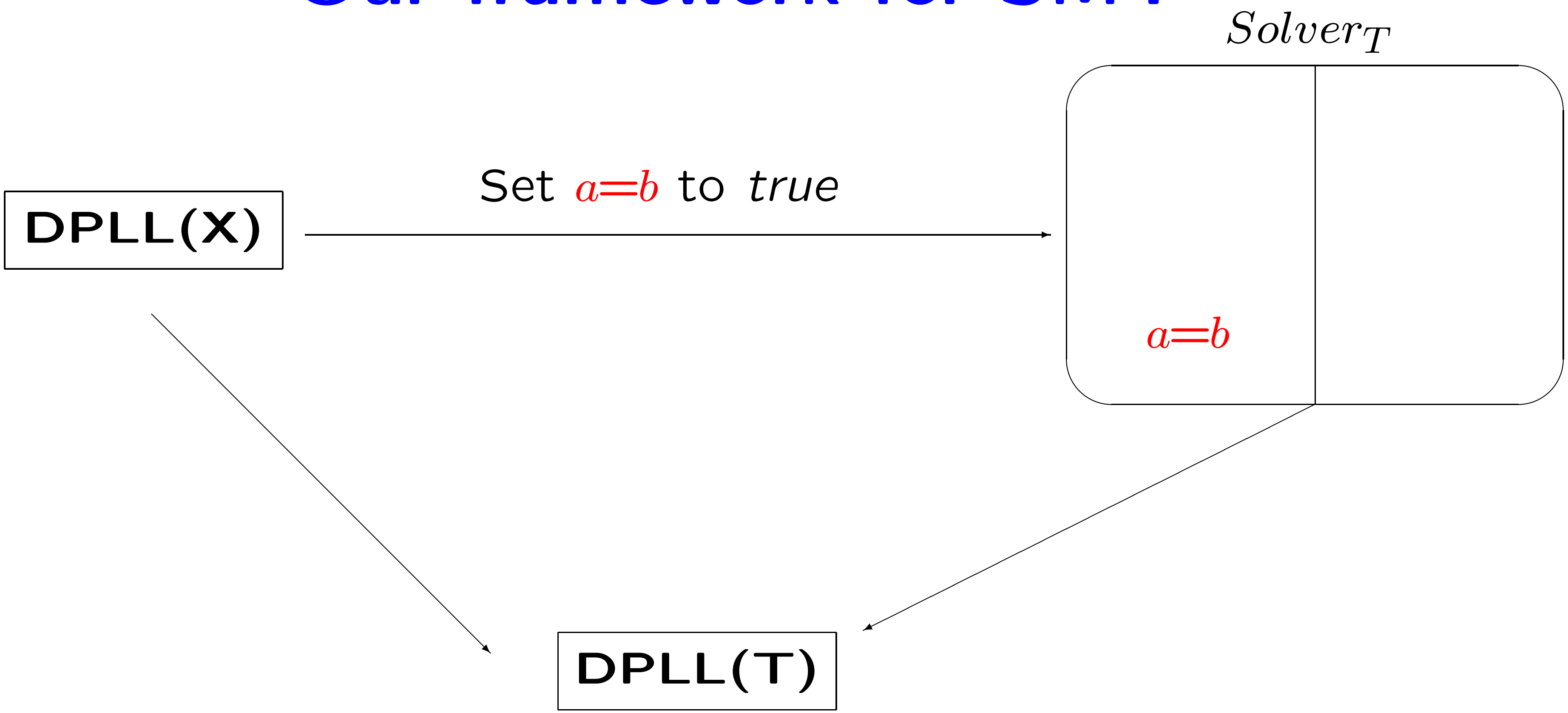
- Advantages:
  - Best SAT solver may be used as is
  - Theory information compiled into the translated formula
- Disadvantages:
  - Loss of formula structure, exponential blowup in size
  - Limited range of application
- Tools: UCLID

Our framework for SMT

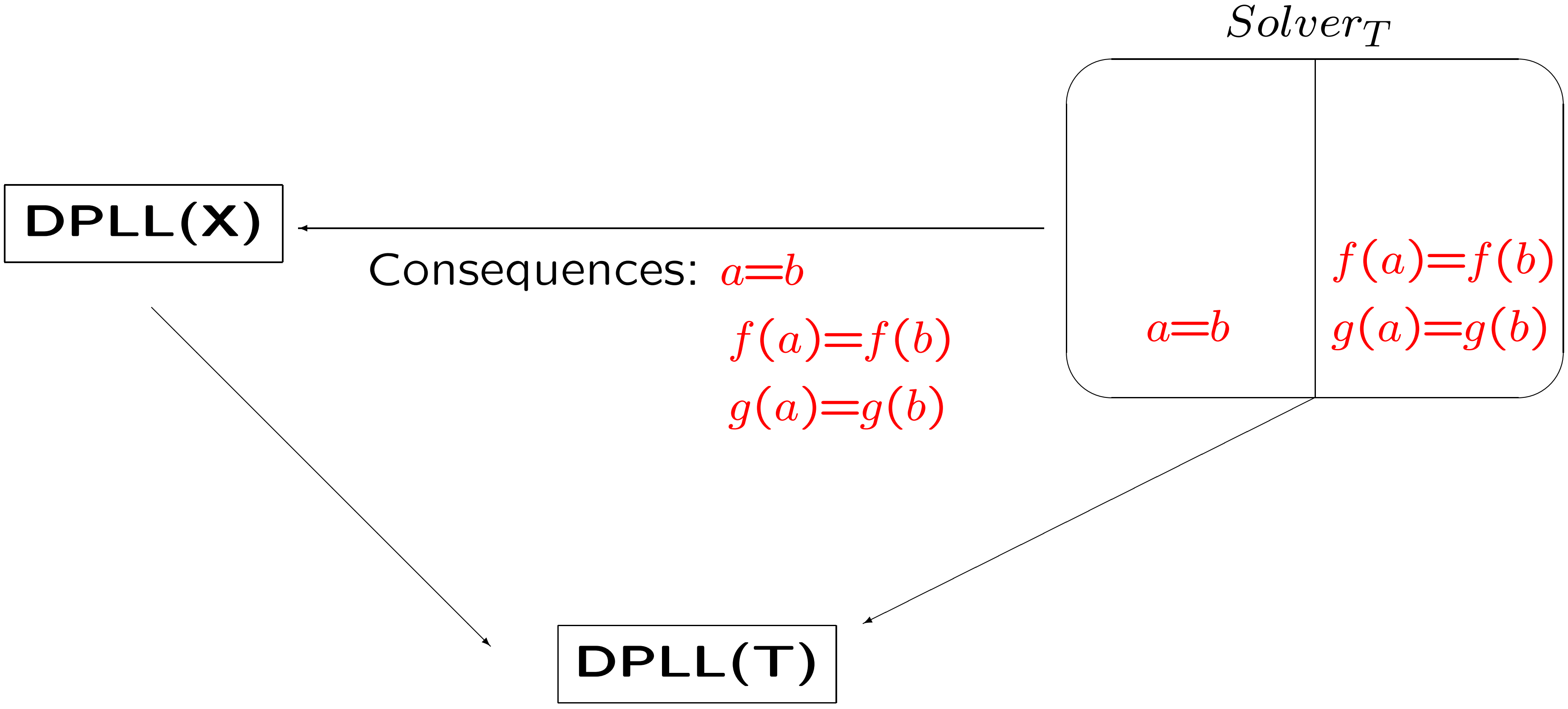


Based on theoretical calculus in [Tinelli'02]

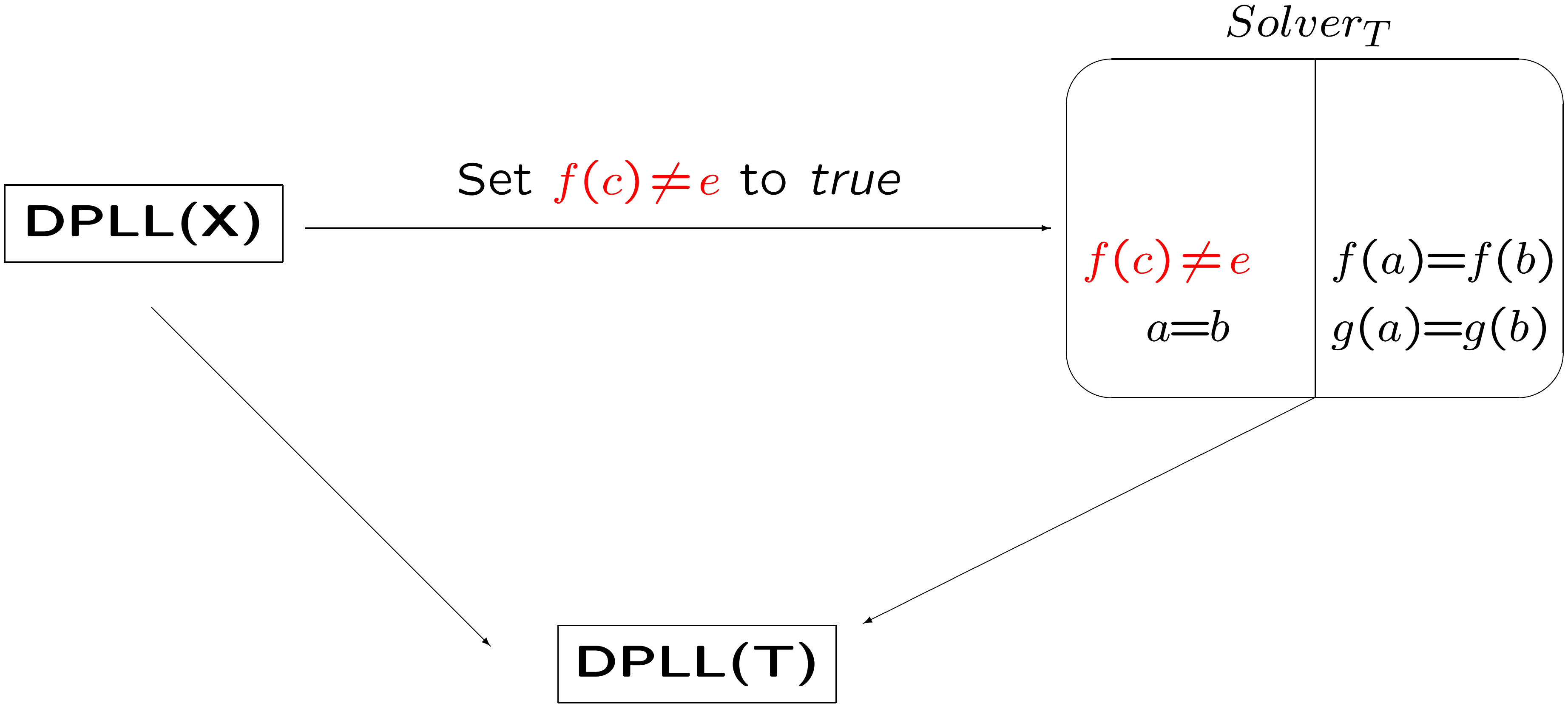
Our framework for SMT



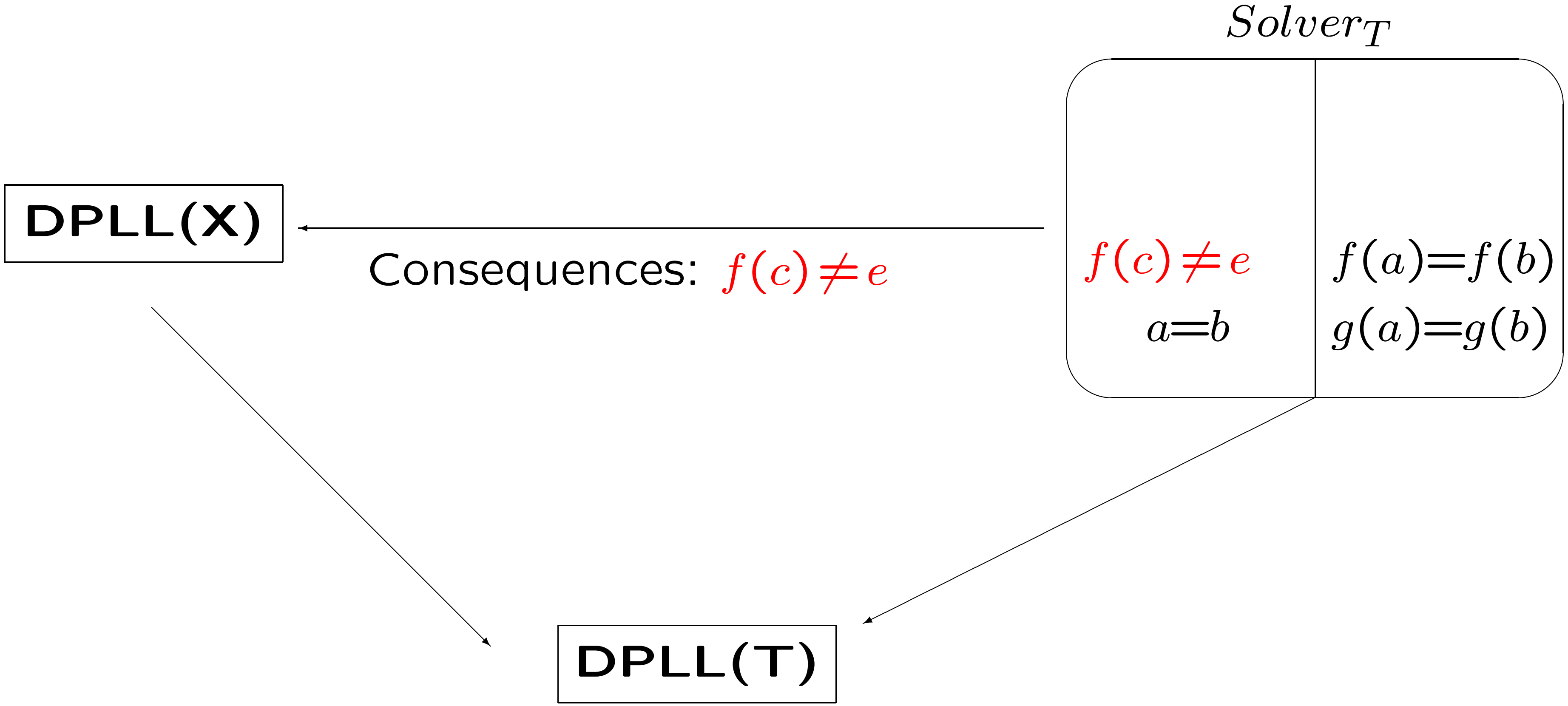
Our framework for SMT



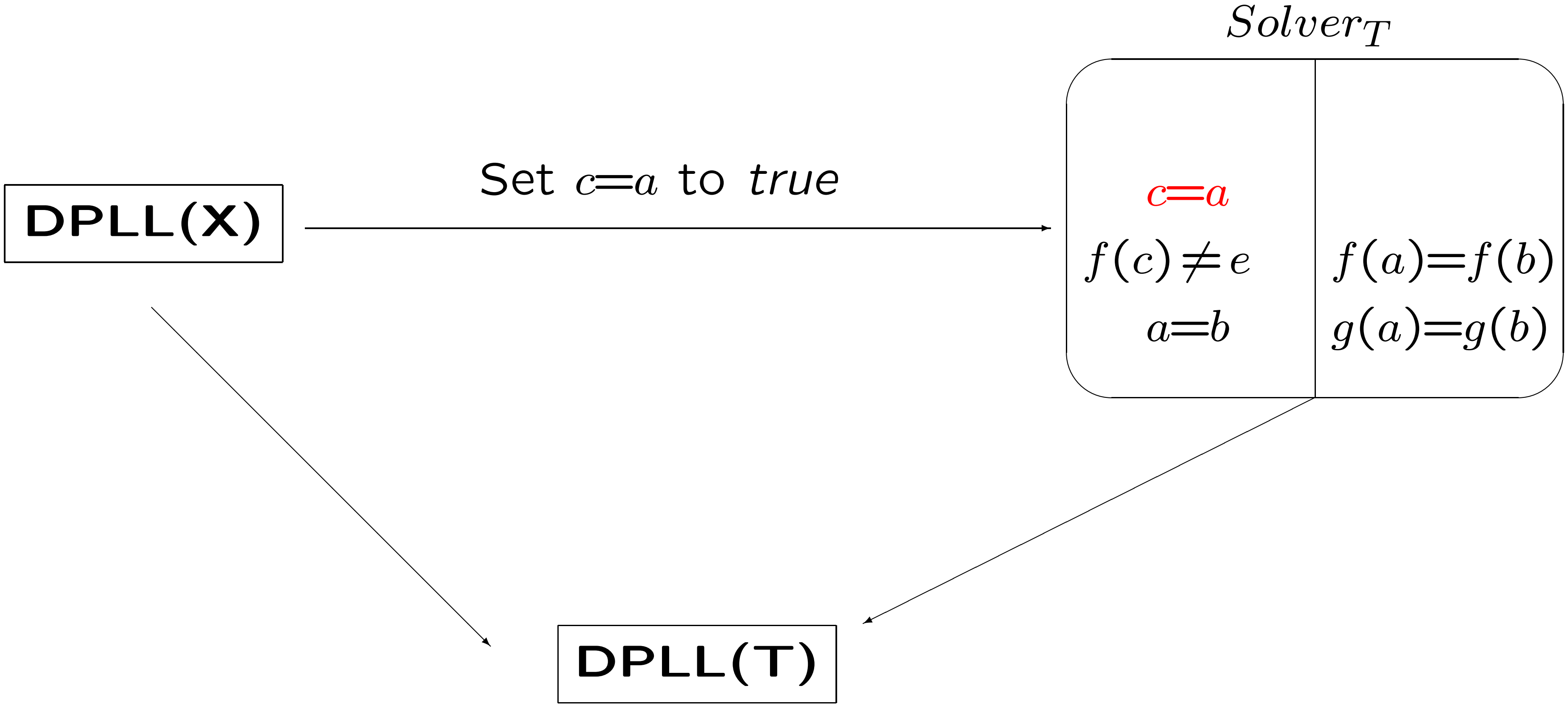
Our framework for SMT



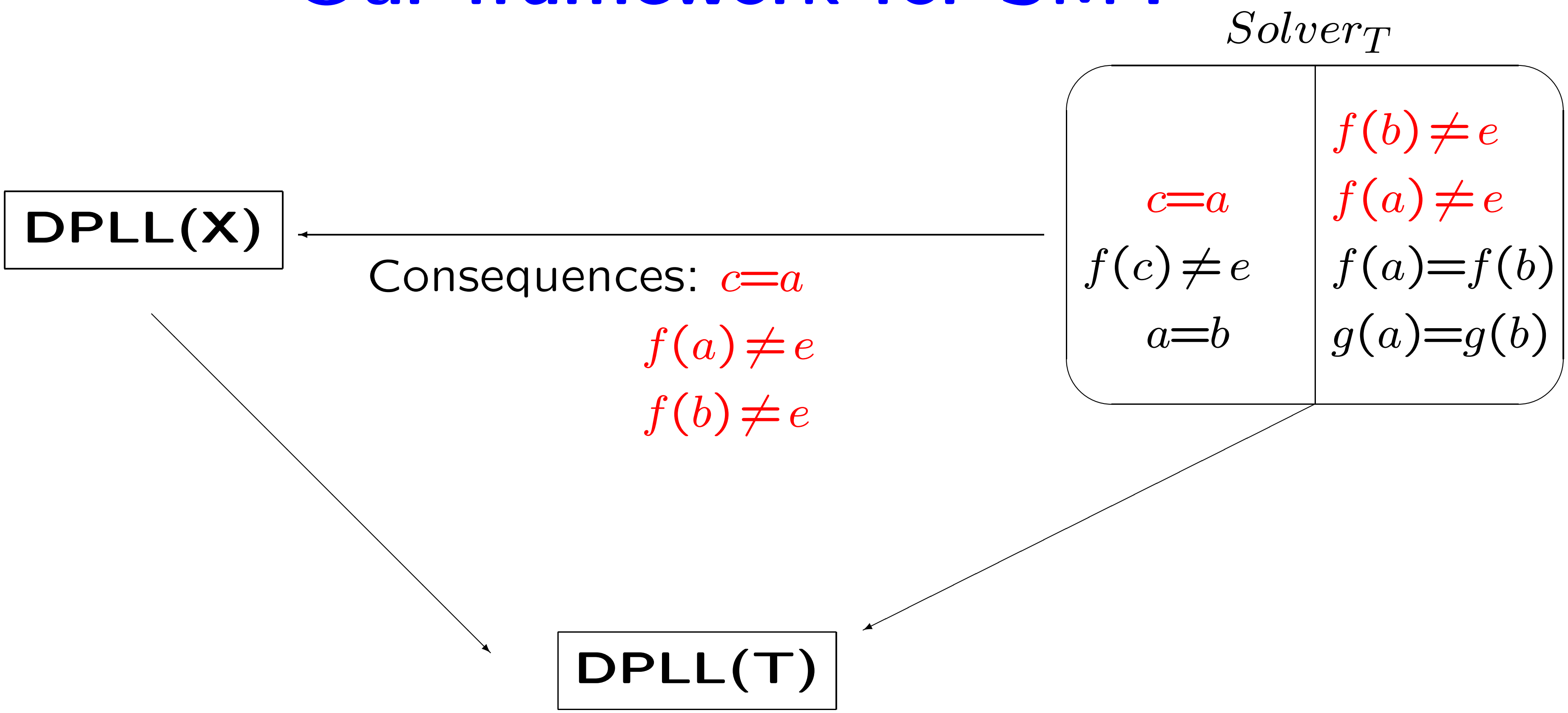
Our framework for SMT



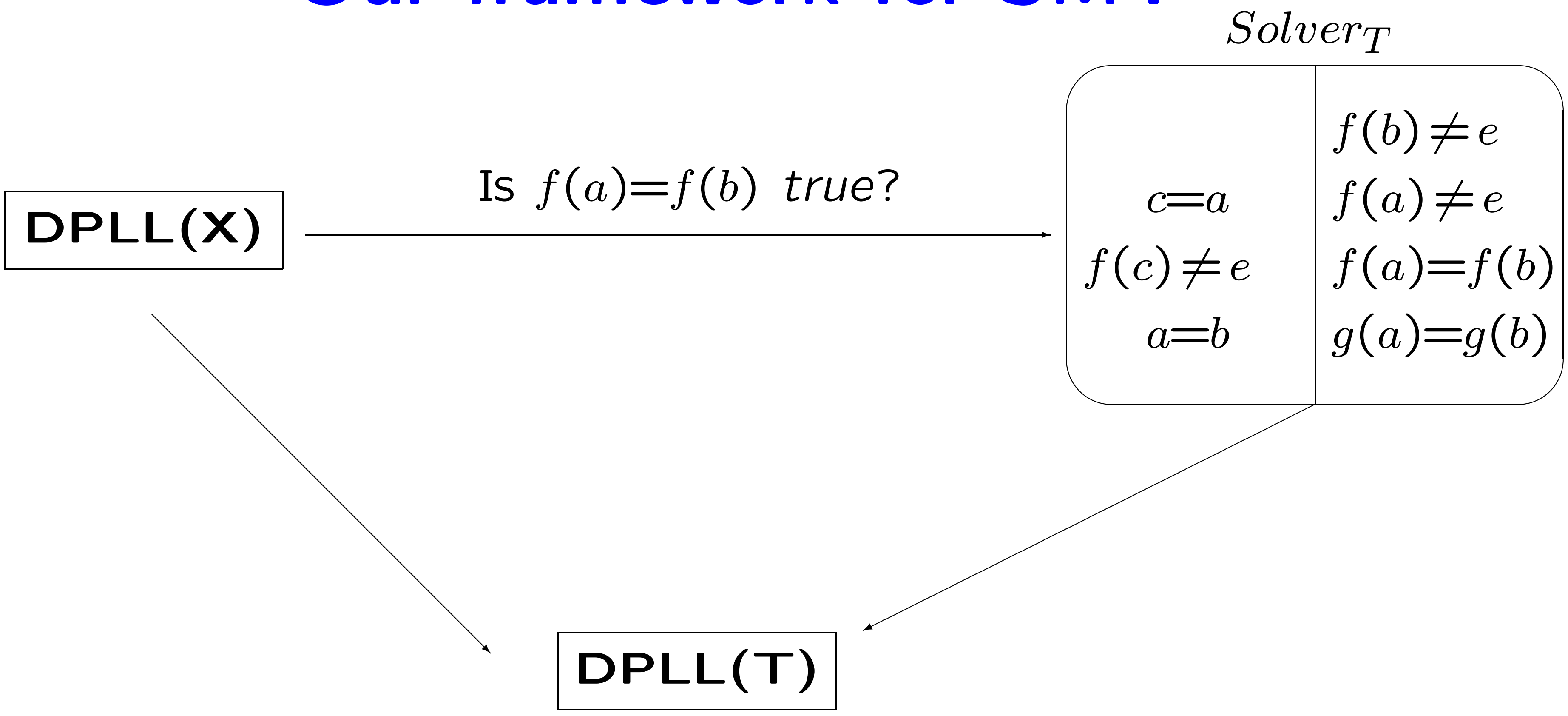
Our framework for SMT



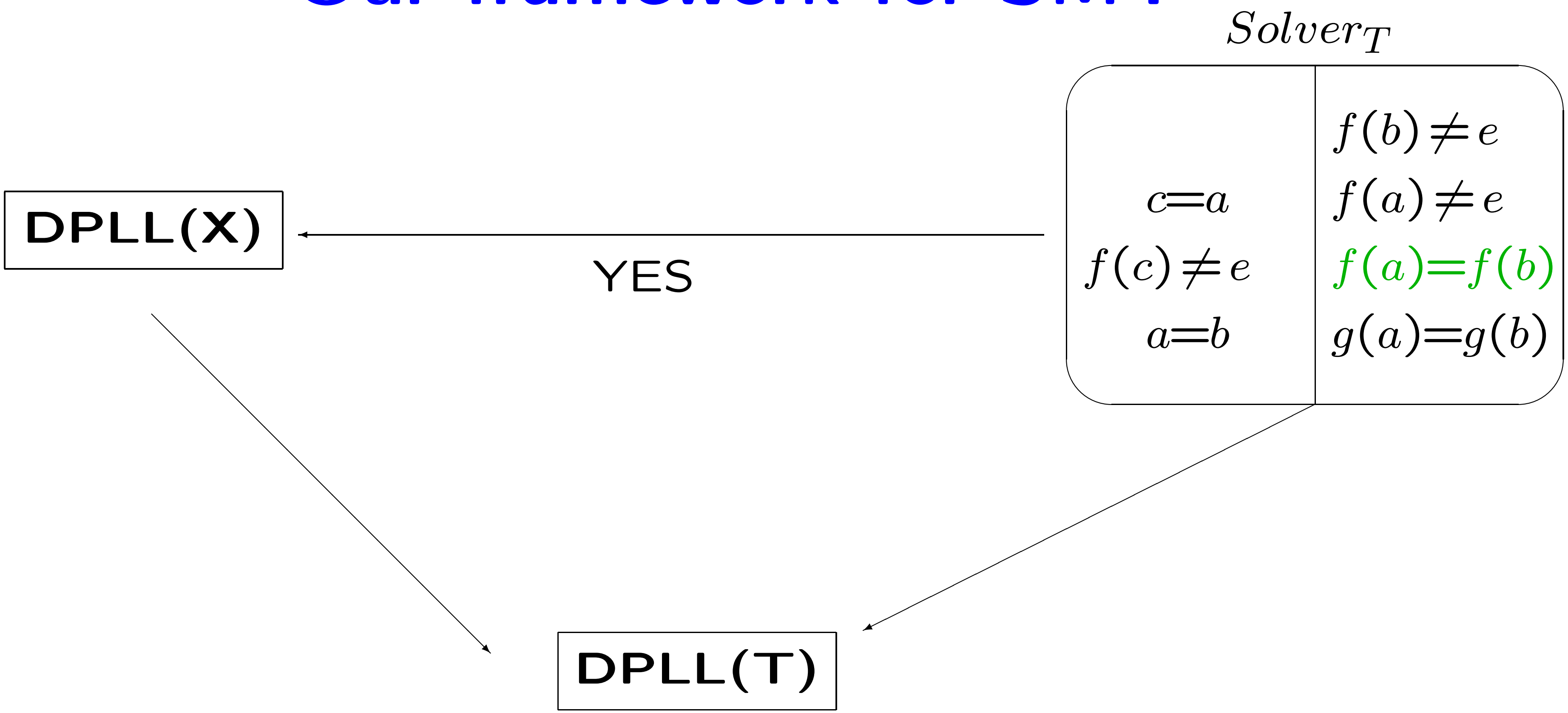
Our framework for SMT



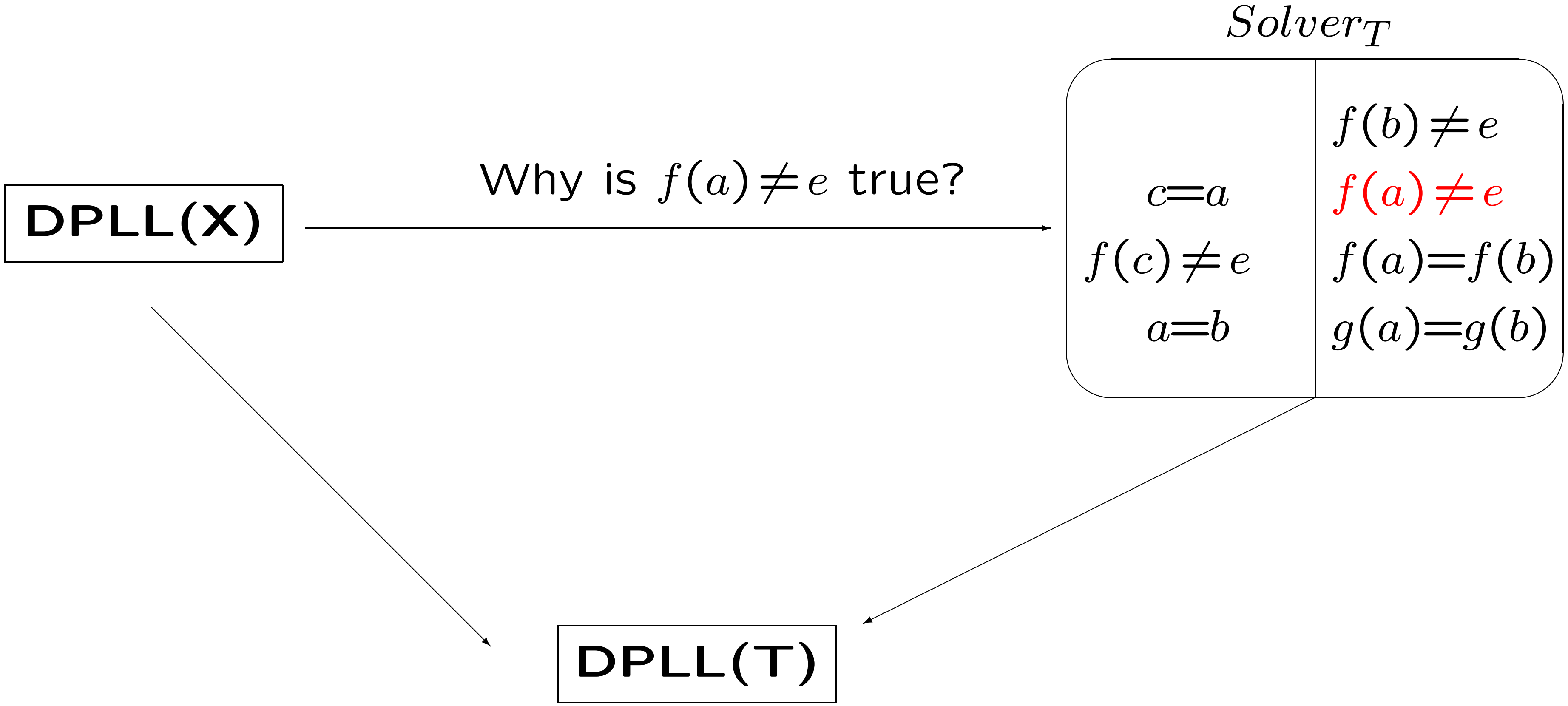
Our framework for SMT



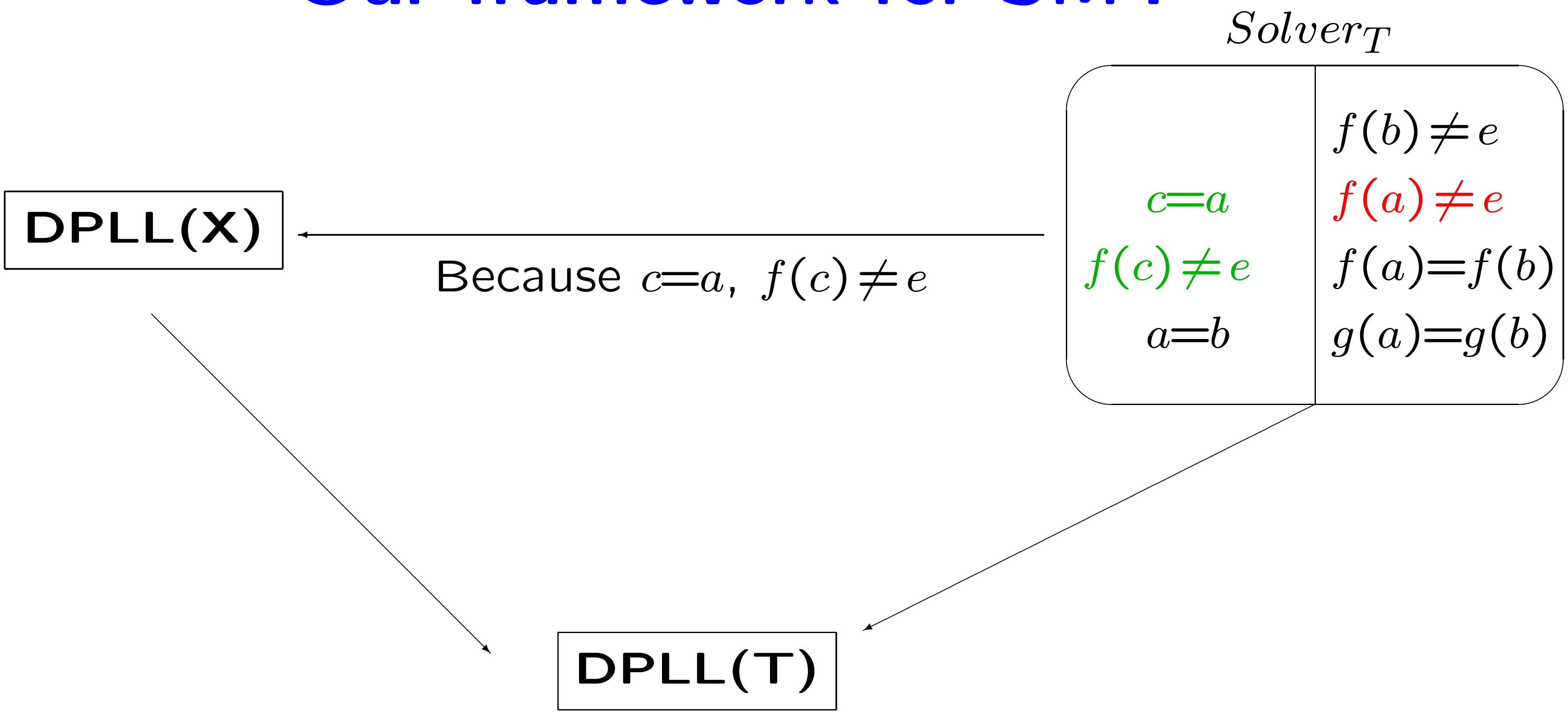
Our framework for SMT



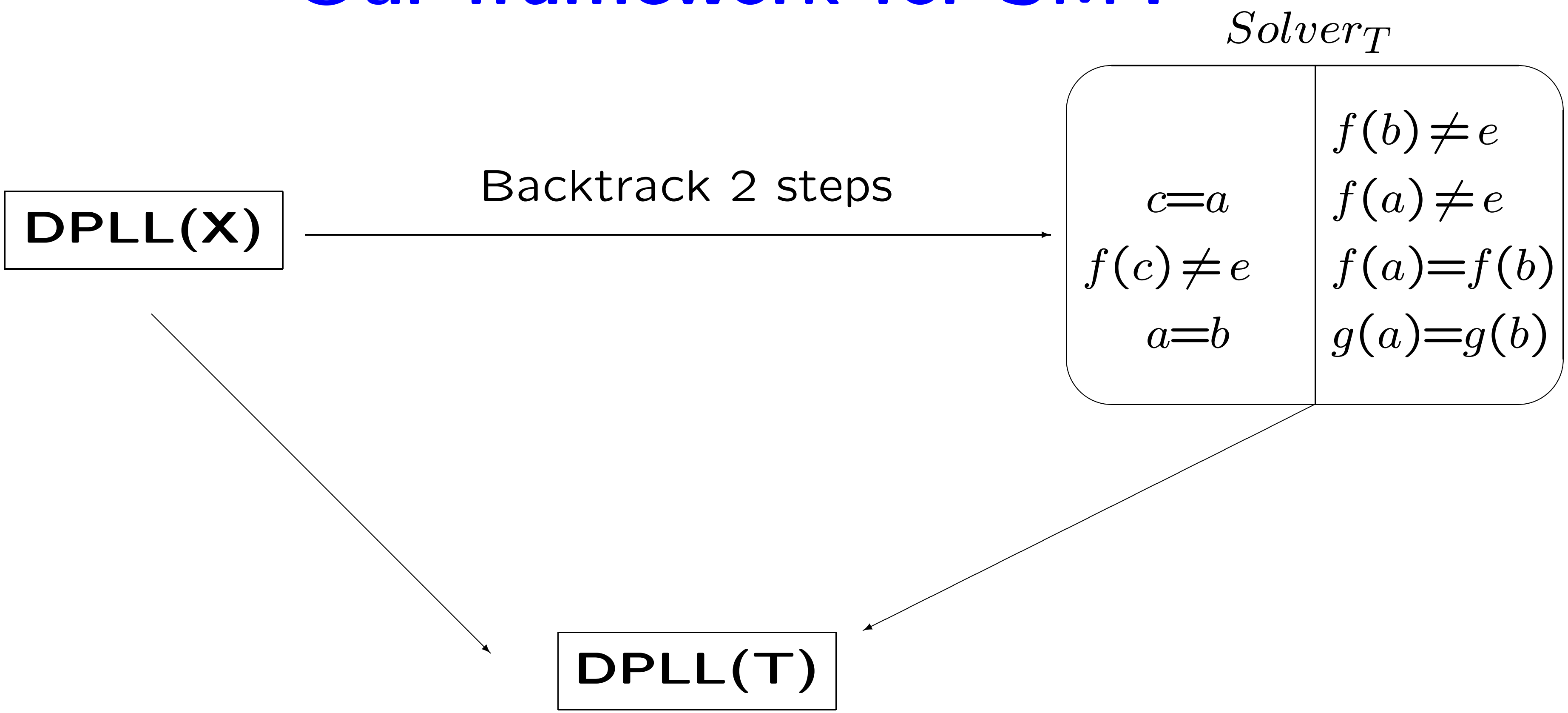
Our framework for SMT



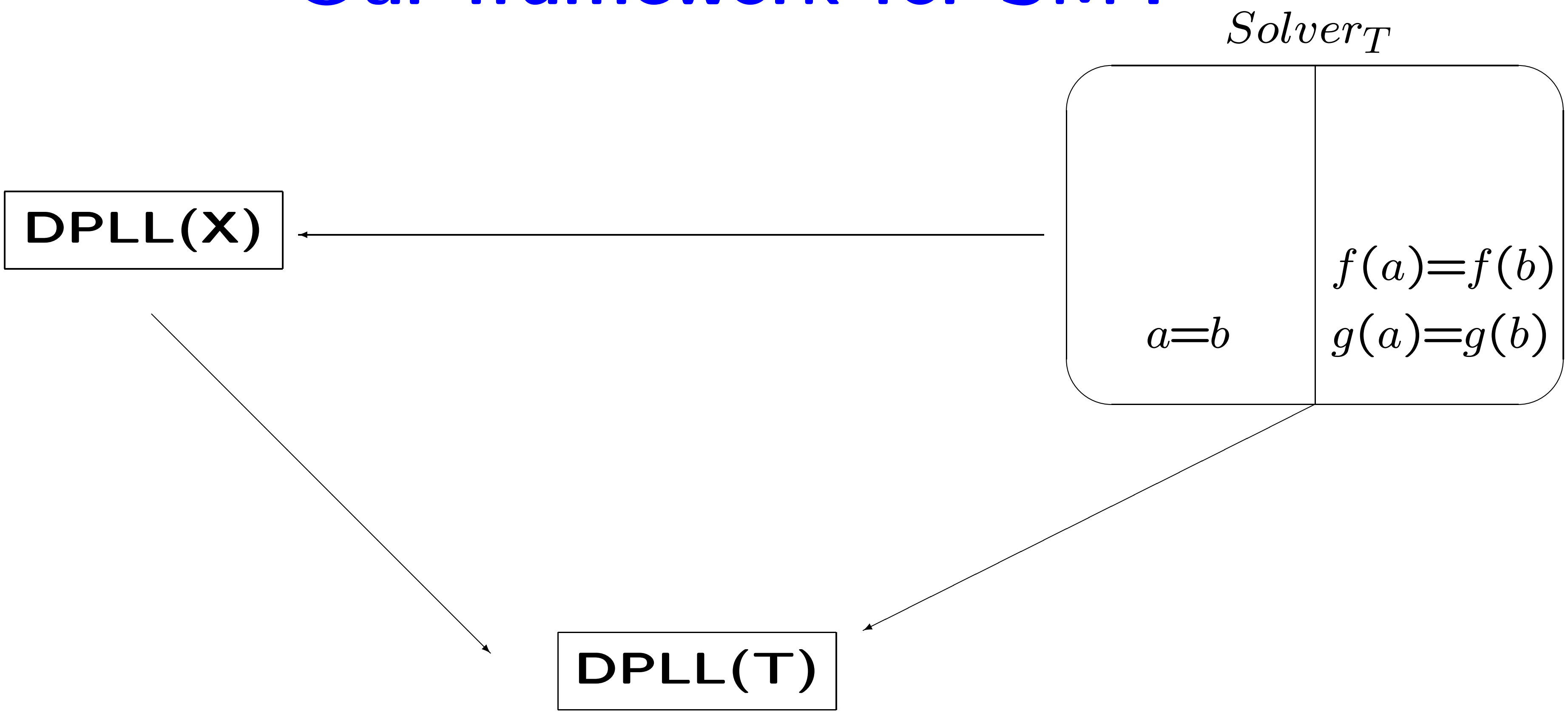
Our framework for SMT



Our framework for SMT



Our framework for SMT



The Davis-Putnam algorithm (DPLL)

- Depth-first search algorithm with backtracking
- At each point, the algorithm keeps a partial interpretation and tries to extend it
- Three successful mechanisms to speed up the search
  - Branching heuristic: determines the literal to extend the interpretation
  - Unit propagation: prunes the search space
  - Conflict Analysis: indicates where to backtrack to and adds lemmas

Branching heuristics

- Unassigned literal with the highest score is selected
- New literals introduced in CNF translation can be selected
- VSIDS heuristic [Moskewicz et al '01]

T-based Branching heuristics

- Unassigned literal with the highest score is selected
- New literals introduced in CNF translation can be selected
- VSIDS heuristic [Moskewicz et al '01]
- Theory-dependent heuristics

Unit Propagation

- A literal appearing in a unit clause has to be true
- EXAMPLE:
  - Consider the binary clause

$$a \neq d \quad \vee \quad g(c)=h(a)$$

  - Now add $a=d$ to the interpretation.
  - The binary clause becomes unit and $g(c)=h(a)$ is added to the interpretation
- State-of-the-art mechanism to detect unit clauses: two watched literal scheme [Moskewicz'01]

T-based Unit Propagation

- A literal appearing in a unit clause has to be true
- EXAMPLE:
  - Consider the binary clause

$$c \neq d \ \vee \ g(c) = h(a)$$

  - Now add $a = d$ to the current interpretation $I = \{a = c\}$.
  - The binary clause becomes unit due to the theory and $g(c) = h(a)$ is added to the interpretation
- Literals returned by SetTrue allow $DPLL$(X) to detect these unit clauses

Conflict analysis

- Analysis performed on the implication graph
- Literals true due to
  - decision (no antecedent in the graph)
  - Unit propagation
- Learning schemes: decision scheme, 1UIP, 2UIP, AllUIP

T-based Conflict analysis

- Analysis performed on the implication graph
- Literals true due to
  - decision (no antecedent in the graph)
  - T-based unit propagation
- Learning schemes similar to decision scheme, 1UIP, 2UIP, AllUIP
- UIP-based learning schemes do not lift with non-exhaustive solvers

Comparison with existing approaches

- Neither loss of structure nor blowup in size
- Theory information used to drive the search
- General framework
- Benefits from improvements in SAT technology

A concrete case: EUF with offsets

- Extension of EUF, but not full CLU
- The sintax is:

$$\begin{aligned} formula :== \;& \mathbf{true} \mid \mathbf{false} \mid predicateSymbol(int\_term, \ldots, int\_term) \\ & \mid \neg formula \mid (formula \vee formula) \\ & \mid (formula \wedge formula) \mid (int\_term = int\_term) \end{aligned}$$

$$\begin{aligned} int\_term :== \;& functionSymbol(int\_term, \ldots, int\_term) \\ & \mid ite(formula, int\_term, int\_term) \\ & \mid succ(int\_term) \mid pred(int\_term) \end{aligned}$$

A solver for EUF with offsets

- New DST-like algorithm for CC with offsets [Nieuwenhuis and Oliveras '03] is the key ingredient
- Two initial transformations at the formula level done once and for all
- After that, only (dis)equalities between constants

A solver for EUF with offsets

The full solver is an extension of the CC algorithm:

- Deals with disequalities
- Incremental and backtrackable
- Explanations based on CC with proof extraction [Nieuwenhuis and Oliveras '04]

Experimental results

Comparison with lazy approaches:

Family	SVC	ICS	DPLL(T)
Buggy Cache	(1 T) 6000	179	7
Code Validation	57	55	4
DLX processor	17	4	1
Elf processor	(1 T) 6078	(4 T) 24001	575
OOO-rf	(2 T) 12666	(2 M) 12458	6385
OOO-tag	(4 T) 28768	(2 M, 2 T) 24050	1979
Load-Store	(3 T) 18475	(1 M, 1 T) 12167	30
Cache Protocol	(4 T) 26112	(5 T) 32022	3601
Two queues	1872	(2 M) 12175	74

T: timeout (more than 6000s.)

M: out of memory, counted as timeout

Experimental results

Comparison with eager approaches (using BerkMin):

Family	SD	Hybrid	DPLL(T)
Buggy Cache	2	3	7
Code Validation	45	28	4
DLX processor	10	13	1
Elf processor	5882	3182	575
OOO-rf	(2 T) 18211	(1 T) 10126	6385
OOO-tag	247	6918	1979
Load-Store	51	45	30
Cache Protocol	4151	209	3601
Two queues	407	793	74

Experimental results

Comparison with eager approaches (using Siege):

Family	SD	Hybrid	DPLL(T)
Buggy Cache	2	4	7
Code Validation	34	28	4
DLX processor	12	13	1
Elf processor	3585	1653	575
OOO-rf	(3 T) 18689	(2 T) 13180	6385
OOO-tag	211	(1 T) 7600	1979
Load-Store	54	45	30
Cache Protocol	4594	228	3601
Two queues	858	(1 T) 6809	74

Conclusions and future work

Conclusions:

- New approach for SMT
- Combines advantages of lazy and eager approaches
- Experimental tests are highly positive

Future work:

- Experiment with more theories
- Define isolated core functionalities of the $DPLL$(X) engine
- Extend to non-quantifier-free formulas