(Yet another) decision procedure for Equality Logic

Ofer Strichman and Orly Meir

Technion

Equality Logic

- ϕ^E: $(\overset{0}{x_1} = \overset{0}{x_2} \wedge (\overset{0}{x_2} = \overset{1}{x_3} \vee \overset{0}{x_1} \neq \overset{1}{x_3}))$

- Domain: $x_1, x_2, x_3 \in \mathbb{N}$

- The satisfiability problem: is there an assignment to x_1, x_2, x_3 that satisfies ϕ^E ?

- Q: When is Equality Logic useful ?...

Equality Logic

- ϕ^E: $\quad \overset{0}{x_1} = \overset{0}{x_2} \wedge (\overset{0}{x_2} = \overset{1}{x_3} \vee \overset{0}{x_1} \neq \overset{1}{x_3})$

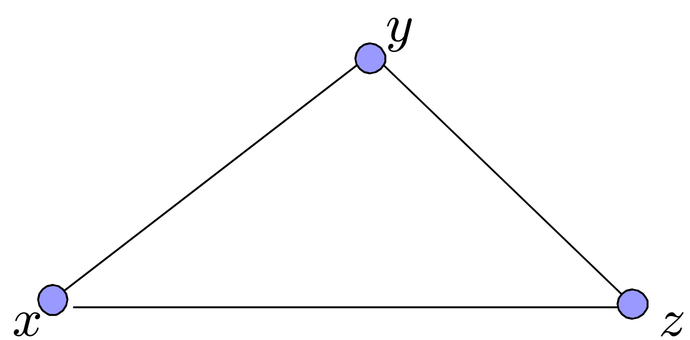
- A: Mainly when combined with Uninterpreted Functions $f(x,y)$, $g(z)$,…

- Mainly used in proving equivalences, but not only.

Basic notions

$$\phi^E: x = y \wedge y = z \wedge z \neq x$$

(non-polar) Equality Graph:



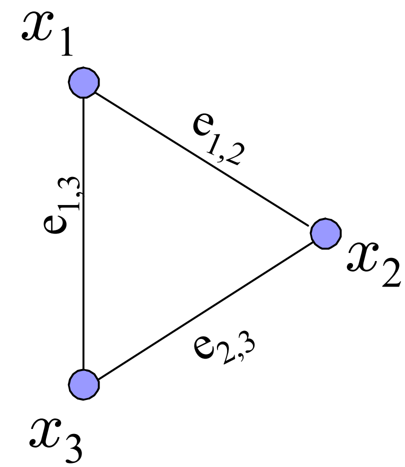
Gives an abstract view of ϕ^E

From Equality to Propositional Logic

Bryant & Velev CAV'00 – the *Sparse* method

$\phi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$

$\mathcal{B}: \quad e_{1,2} \quad \wedge \quad e_{2,3} \quad \wedge \quad \neg e_{1,3}$



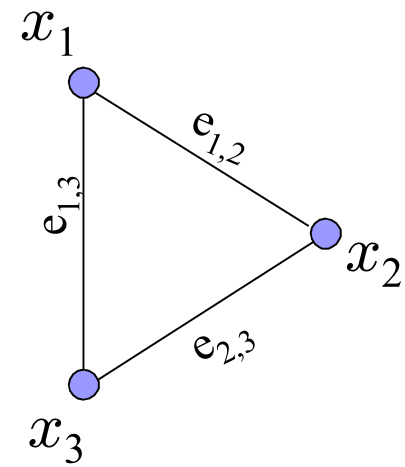
- Encode all edges with Boolean variables
  - This is an abstraction
  - Transitivity of equality is lost!
  - Must add transitivity constraints!

From Equality to Propositional Logic

Bryant & Velev CAV'00 – the *Sparse* method

$\phi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$

$\mathcal{B}: \quad e_{1,2} \quad \wedge \quad e_{2,3} \quad \wedge \quad \neg e_{1,3}$



- Transitivity Constraints: For each cycle of size n, forbid a true assignment to n-1 edges

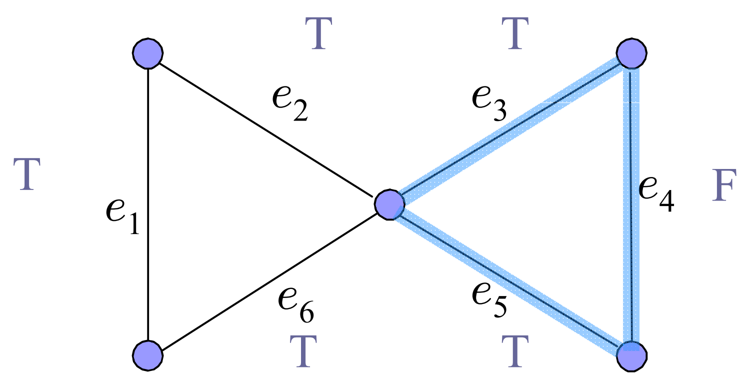
$$\mathcal{T}^S = (e_{1,2} \wedge e_{2,3} \rightarrow e_{1,3}) \wedge$$
$$(e_{1,2} \wedge e_{1,3} \rightarrow e_{2,3}) \wedge$$
$$(e_{1,3} \wedge e_{2,3} \rightarrow e_{1,2})$$

Check: $\mathcal{B} \wedge \mathcal{T}^S$

From Equality to Propositional Logic

Bryant & Velev CAV’00 – the *Sparse* method

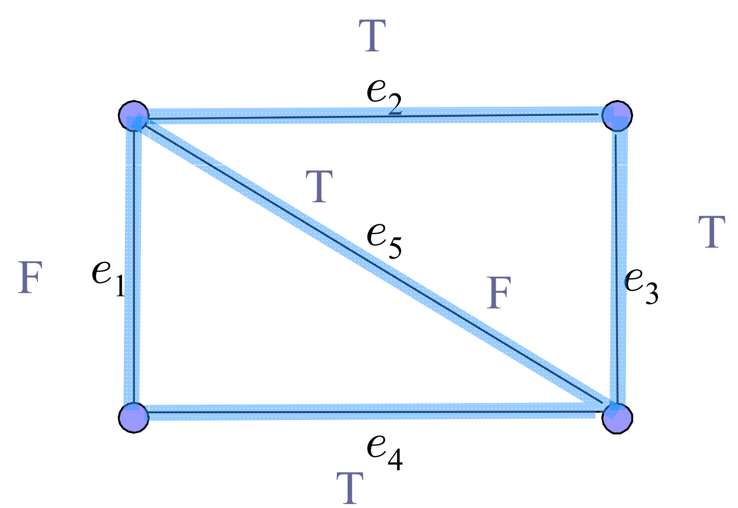
- Thm-1: It is sufficient to constrain simple cycles only



From Equality to Propositional Logic

Bryant & Velev CAV'00 – the *Sparse* method

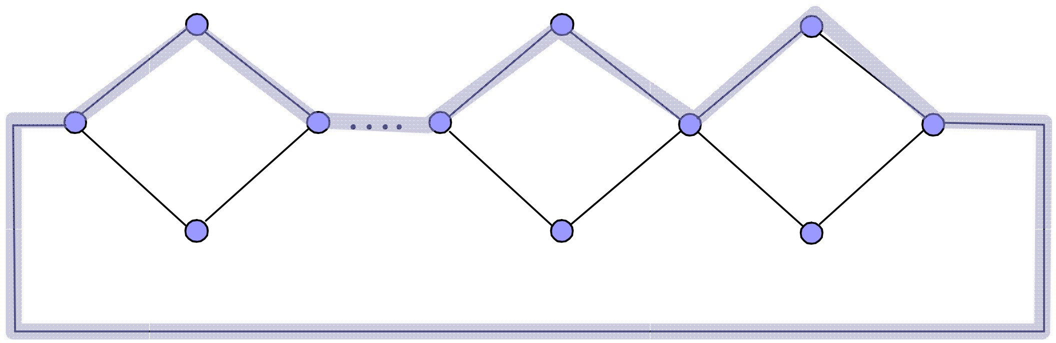
- Thm-2: It is sufficient to constrain chord-free simple cycles



From Equality to Propositional Logic

Bryant & Velev CAV'00 – the *Sparse* method

- Still, there can be an exponential number of chord-free simple cycles…

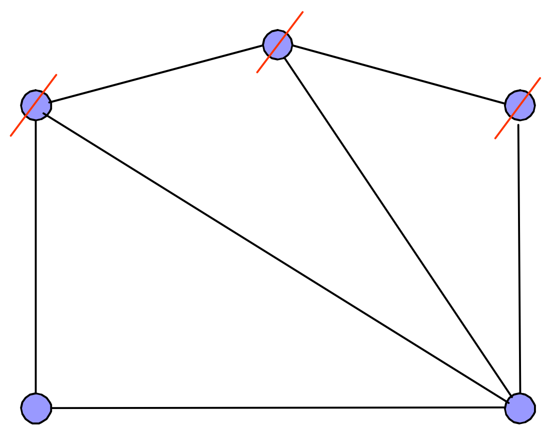


- Solution: make the graph 'chordal'!

From Equality to Propositional Logic

Bryant & Velev CAV'00 – the *Sparse* method

- Dfn: A graph is chordal iff every cycle of size 4 or more has a chord.
- How to make a graph chordal ? eliminate vertices one at a time, and connect their neighbors.

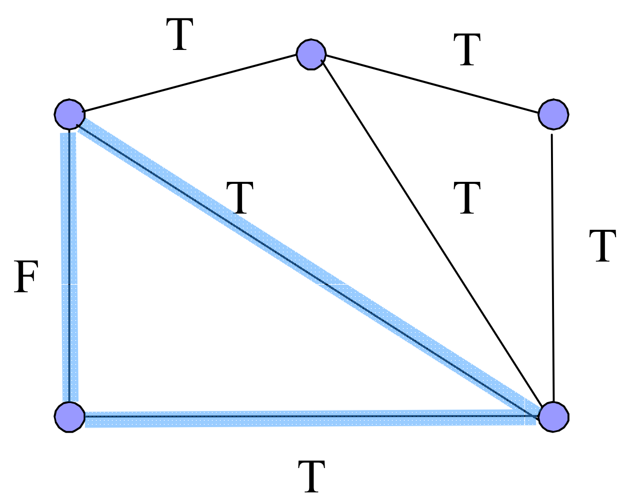


From Equality to Propositional Logic

Bryant & Velev CAV’00 – the *Sparse* method

- In a chordal graph, it is sufficient to constrain only triangles.

Contradiction!



- Polynomial # of edges and constraints.
- # constraints = 3 $\times$ #triangles

An improvement

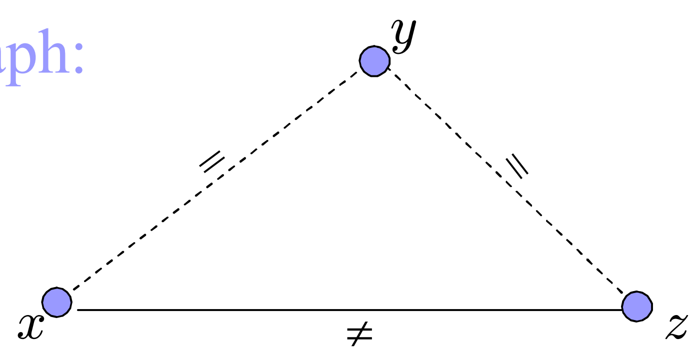
Reduced Transitivity Constraints (RTC)

- So far we did not consider the polarity of the edges.

$$\phi^E: x = y \land y = z \land z \neq x$$

- Assuming ϕ^E is in Negation Normal Form

(polar) Equality Graph:



Monotonicity of NNF

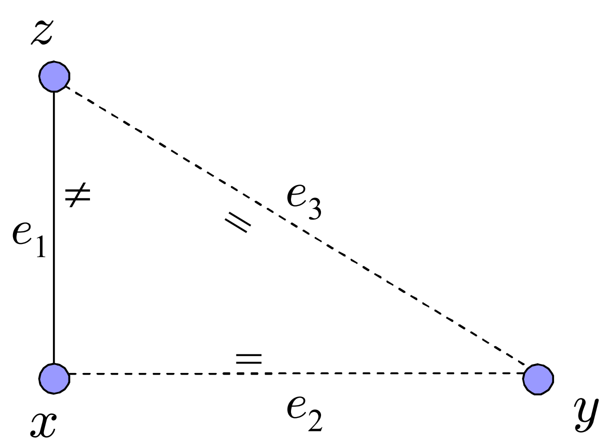
- Thm-3: NNF formulas are *monotonically satisfied* (in CNF this is simply the pure literal rule)

- Let ϕ be in NNF and satisfiable. Thm-3 implies:
  - Let $\alpha \models \phi$
  - Derive α' from α by switching the value of a 'mis-assigned' pure literal in α
  - Now $\alpha' \models \phi$

An improvement

Reduced Transitivity Constraints (RTC)

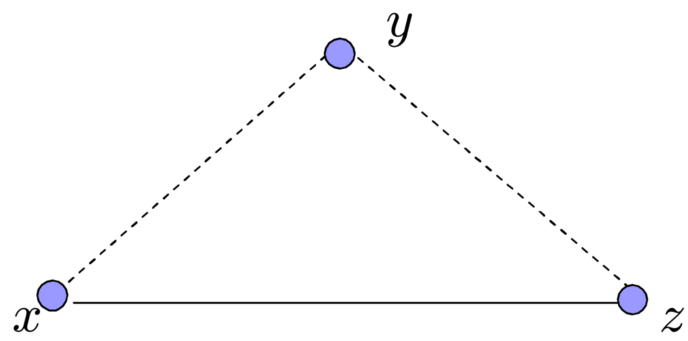
- Claim: in the following graph $\mathcal{T}^{\mathrm{R}} = e_3 \wedge e_2 \rightarrow e_1$ is sufficient



Allowing e.g.
e_1=e_2=T, e_3= F

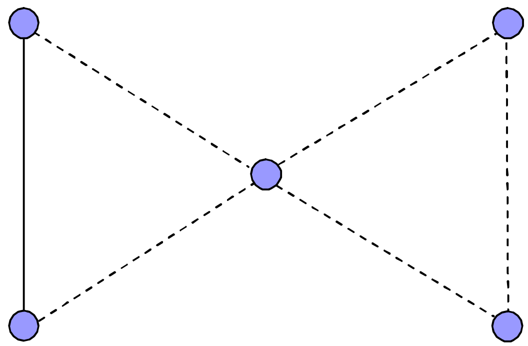
- This is only true because of monotonicity of NNF (an extension of the pure literal rule)

Basic notions



- *Equality Path*: a path made of equalities.
  we write $x =^* z$
- *Disequality Path*: a path made of equalities and exactly one disequality. We write $x \neq^* y$
- *Contradictory Cycle*: two nodes x and y, s.t. $x =^* y$ and $x \neq^* y$ form a contradictory cycle

Basic notions

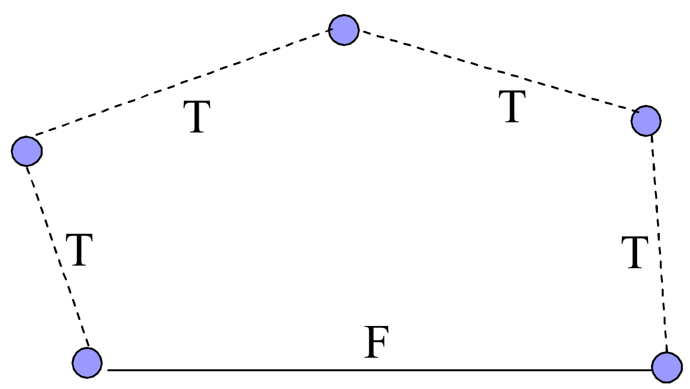


- Thm-4: Every contradictory cycle is either simple or contains a simple contradictory cycle

Definitions

- Dfn: A contradictory Cycle C is *constrained under* $\mathcal{T}$ if $\mathcal{T}$ does not allow this assignment

$C =$



Main theorem

- If

  $\mathcal{T}^{R}$ constrains all simple contradictory cycles,

  and

  For every assignment α^{S}, $\alpha^{S} \models \mathcal{T}^{S} \rightarrow \alpha^{S} \models \mathcal{T}^{R}$

  From the Sparse method

- then

  ϕ^{E} is satisfiable iff $\mathcal{B} \wedge \mathcal{T}^{R}$ is satisfiable

  The Equality Formula

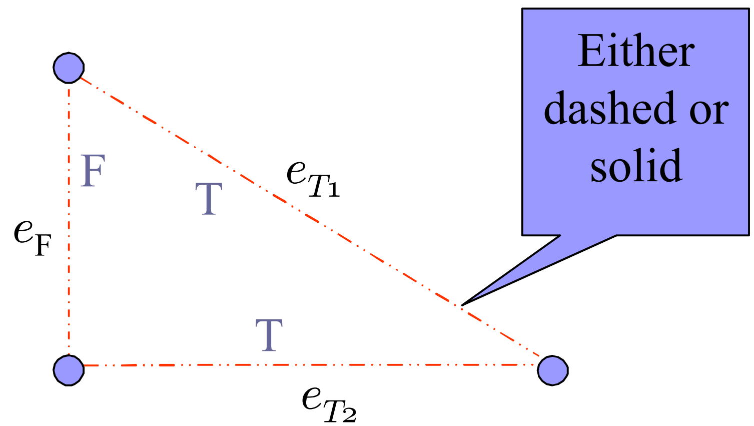
Proof of the main theorem

- (➔) ϕ^E is satisfiable → $\mathcal{B}\wedge\mathcal{T}^S$ is satisfiable → $\mathcal{B}\wedge\mathcal{T}^R$ is satisfiable
- (🡐) Proof strategy:
  - Let α^R be a satisfying assignment to $\mathcal{B} \wedge \mathcal{T}^R$
  - We will construct α^S that satisfies $\mathcal{B} \wedge \mathcal{T}^S$
  - From this we will conclude that ϕ^E is satisfiable

Skip proof

Definitions for the proof…

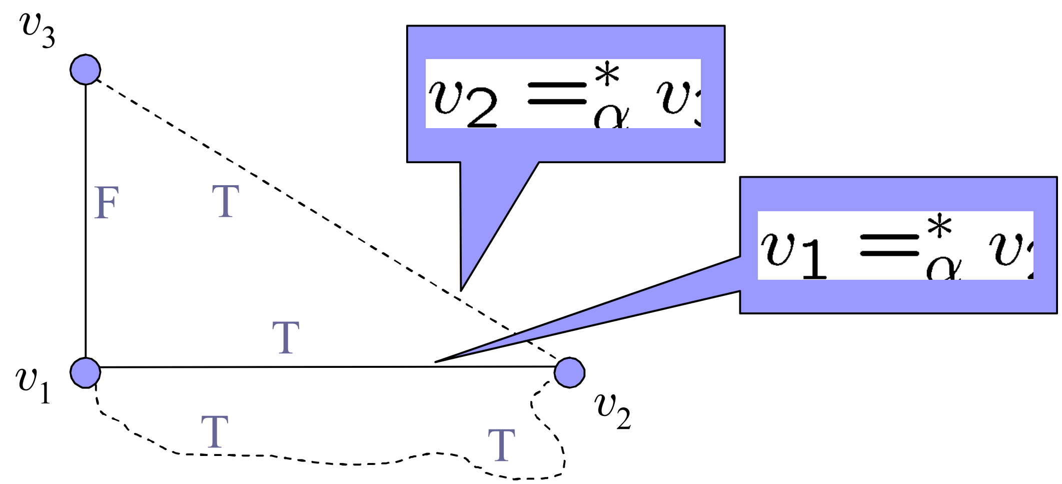
- A *Violating cycle* under an assignment α^{R}:



- This assignment violates $\mathcal{T}^{S}$ but not necessarily $\mathcal{T}^{R}$

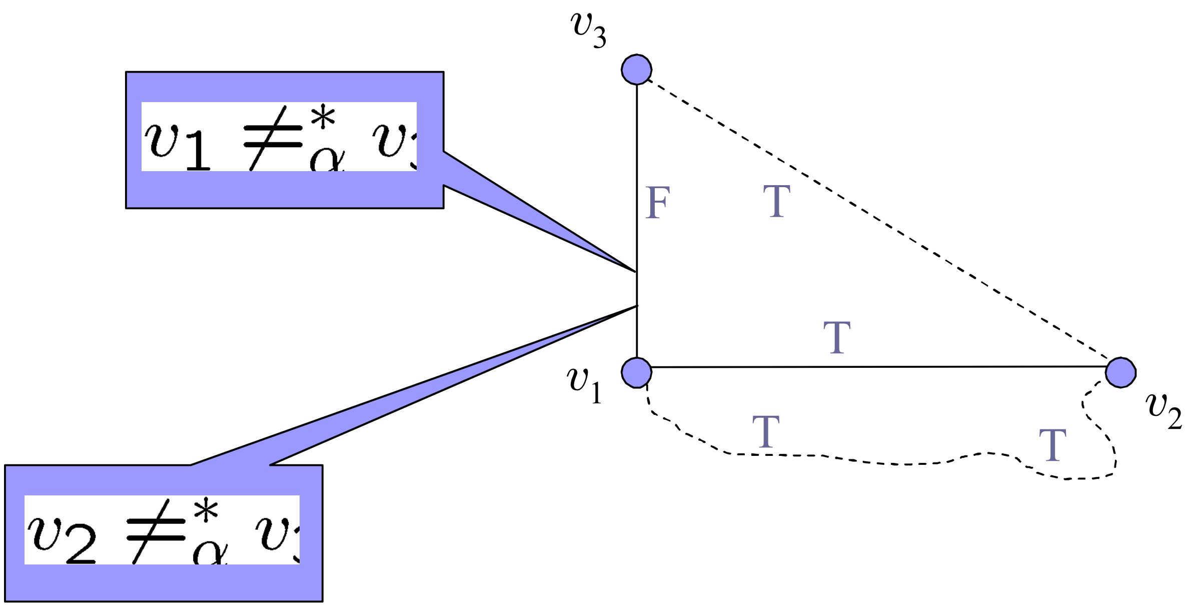
More definitions for the proof…

- An edge $e = (v_i, v_j)$ is ***equal*** *under an assignment* α iff there is an equality path between v_i and v_j all assigned T under α. *Denote*: $v_i =^*_\alpha v$



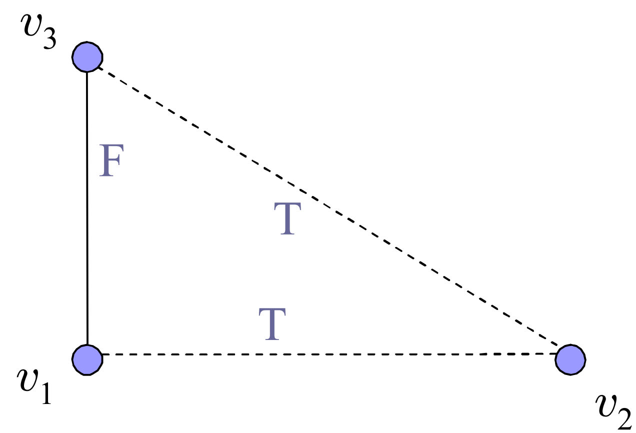
More definitions for the proof…

- An edge $e = (v_i, v_j)$ is ***disequal*** *under an assignment* α iff there is a disequality path between v_i and v_j in which the solid edge is the only one assigned false by α. *Denote:* $v_i \neq^*_\alpha v$



Proof…

- Observation 1:
  The combination $v_1 \neq^*_\alpha v_3 \quad v_1 =^*_\alpha v_2 \quad v_2 =^*_\alpha$
  is impossible if $\alpha = \alpha^R$
  (recall: $\alpha^R \models \mathcal{T}^R$)

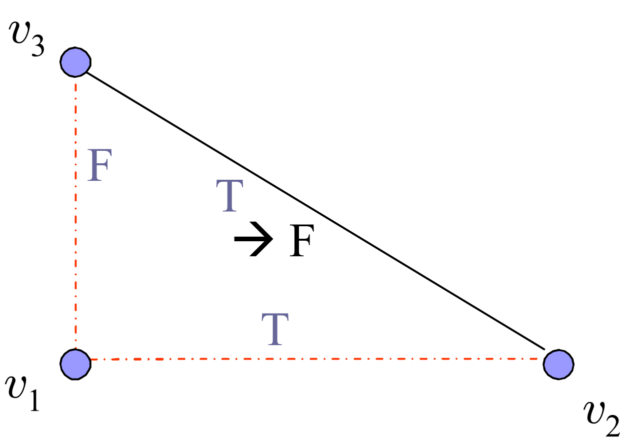


- Observation 2: if (v_1, v_3) is solid, then $v_1 \neq^*_\alpha v$

ReConstructing α^S

<u>Type 1:</u>

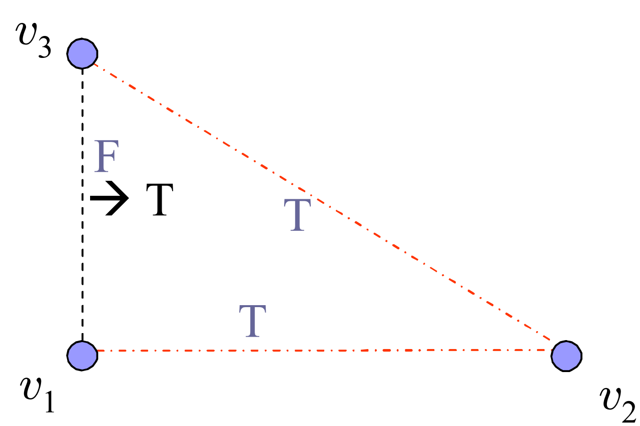
It is *not* the case that $v_2 =_\alpha^* v_3$



- Assign $\alpha^S(e_{23}) = F$

<u>Type 2:</u>

Otherwise it is *not* the case that $v_1 \neq_\alpha^* v_3$



- Assign $\alpha(e_{13}) = T$

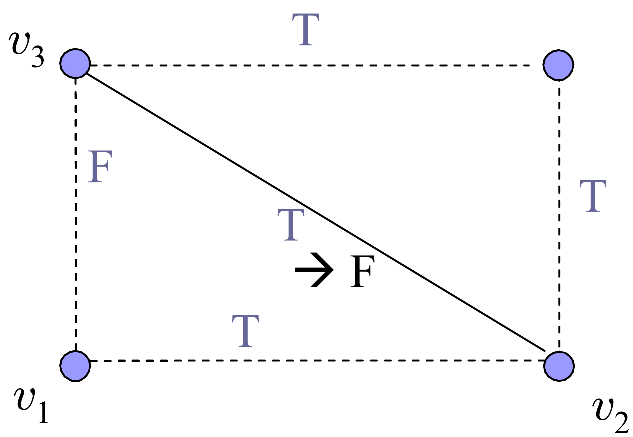
In all other cases $\alpha^S = \alpha^R$

ReConstructing α^S

- Starting from α^R, repeat until convergence:
  - $\alpha(e_T)$:= F in all Type 1 cycles
  - $\alpha(e_F)$:= T in all Type 2 cycles
- All Type 1 and Type 2 triangles now satisfy $\mathcal{T}^S$
- $\mathcal{B}$ is still satisfied (monotonicity of NNF)
- Left to prove: all contradictory cycles are still satisfied

Proof…

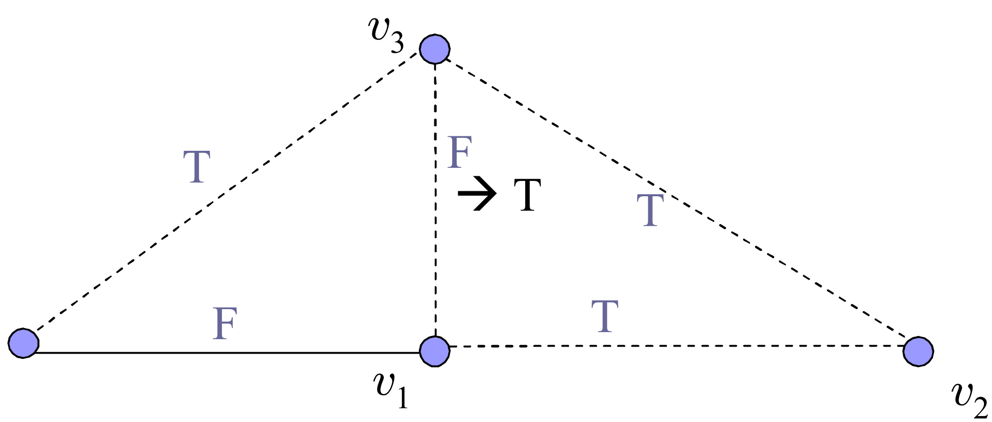
- Invariant: contradictory cycles are not violating throughout the reconstruction.



- $v_2 =^*_\alpha v_3$ contradicts the precondition to make this assignment…

Proof…

- Invariant: contradictory cycles are not violating throughout the reconstruction.

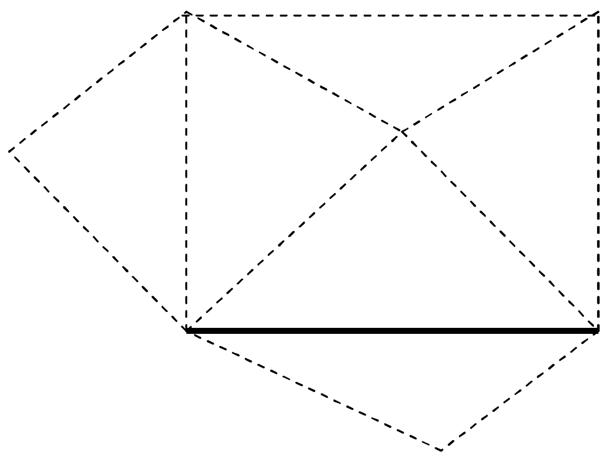


- $v_1 \neq^*_\alpha v$ contradicts the precondition to make this assignment…

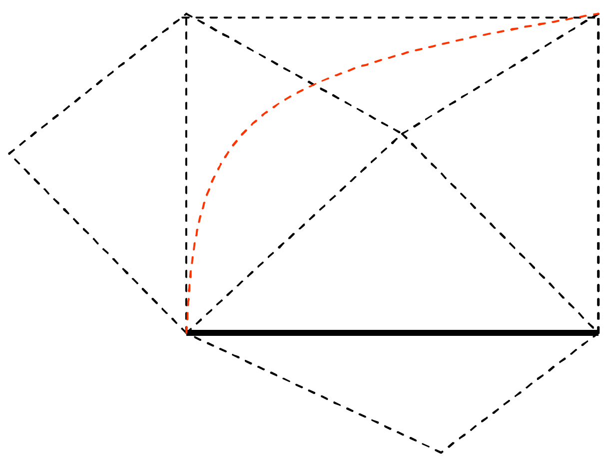
Applying RTC

- How can we use the theorem without enumerating contradictory cycles ?
- Answer:
  - Consider the chordal graph.
  - Constrain triangles if they are part of a (simple) contradictory cycle
  - How?

- Focus on Bi-connected dashed components built on top of a solid edge
  - Includes all contradictory cycles involving this edge

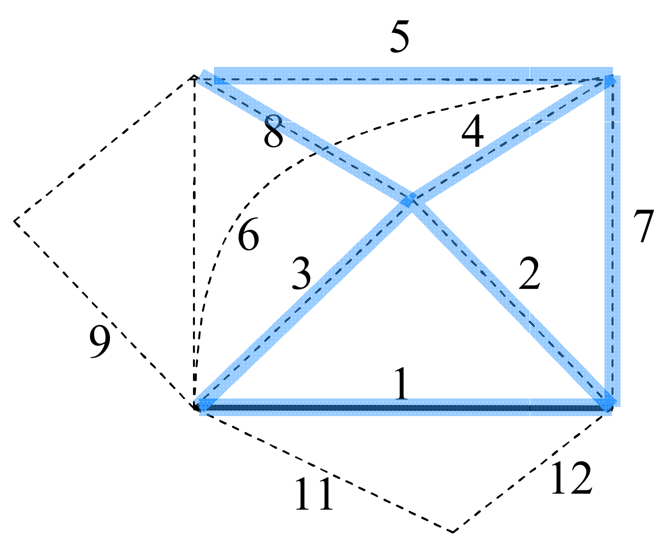


- Make the component chordal
  - Chordal-ity guarantees: every cycle contains a simplicial vertex, i.e. a vertex that its neighbors are connected.



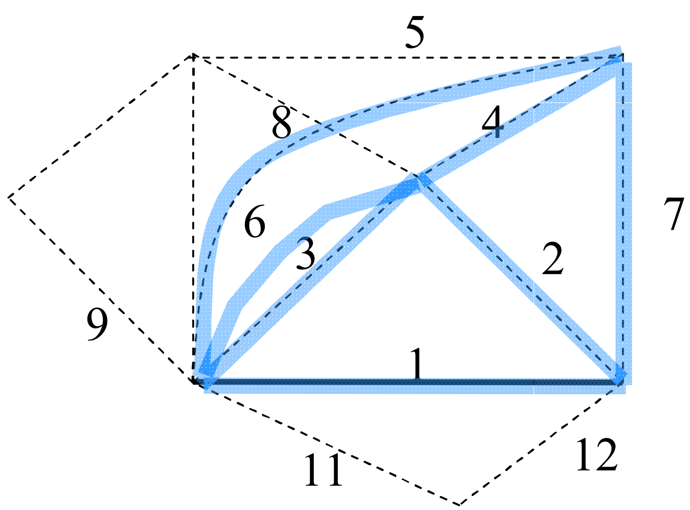
The RTC algorithm

- Constraints cache:
  - $e_2 \wedge e_3 \rightarrow e_1$
  - $e_4 \wedge e_7 \rightarrow e_2$
  - $e_5 \wedge e_8 \rightarrow e_4$



Constrains *all* contradictory cycles

- Constraints cache:
  - $e_2 \wedge e_3 \rightarrow e_1$
  - $e_4 \wedge e_7 \rightarrow e_2$
  - $e_6 \wedge e_3 \rightarrow e_4$



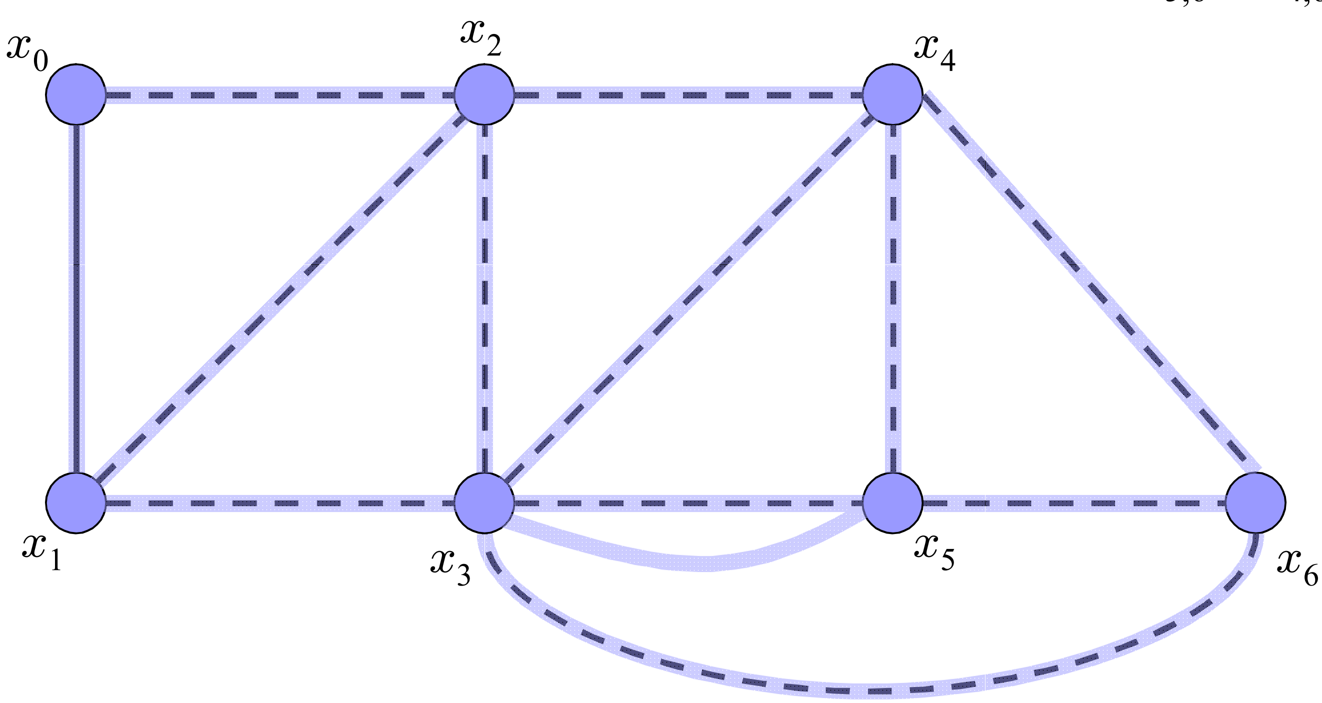
Constraining *simple* contradictory cycles

The constraint $e_{3,6} \wedge e_{3,5} \rightarrow e_{5,6}$ is not added

cache:

...

$e_{5,6} \wedge e_{4,6} \rightarrow e_{4,5}$



Open problem: constrain simple contradictory cycles in P time

Constraining *simple* contradictory cycles

the constraint $e_{3,6} \wedge e_{3,5} \rightarrow e_{5,6}$ is not added, though needed

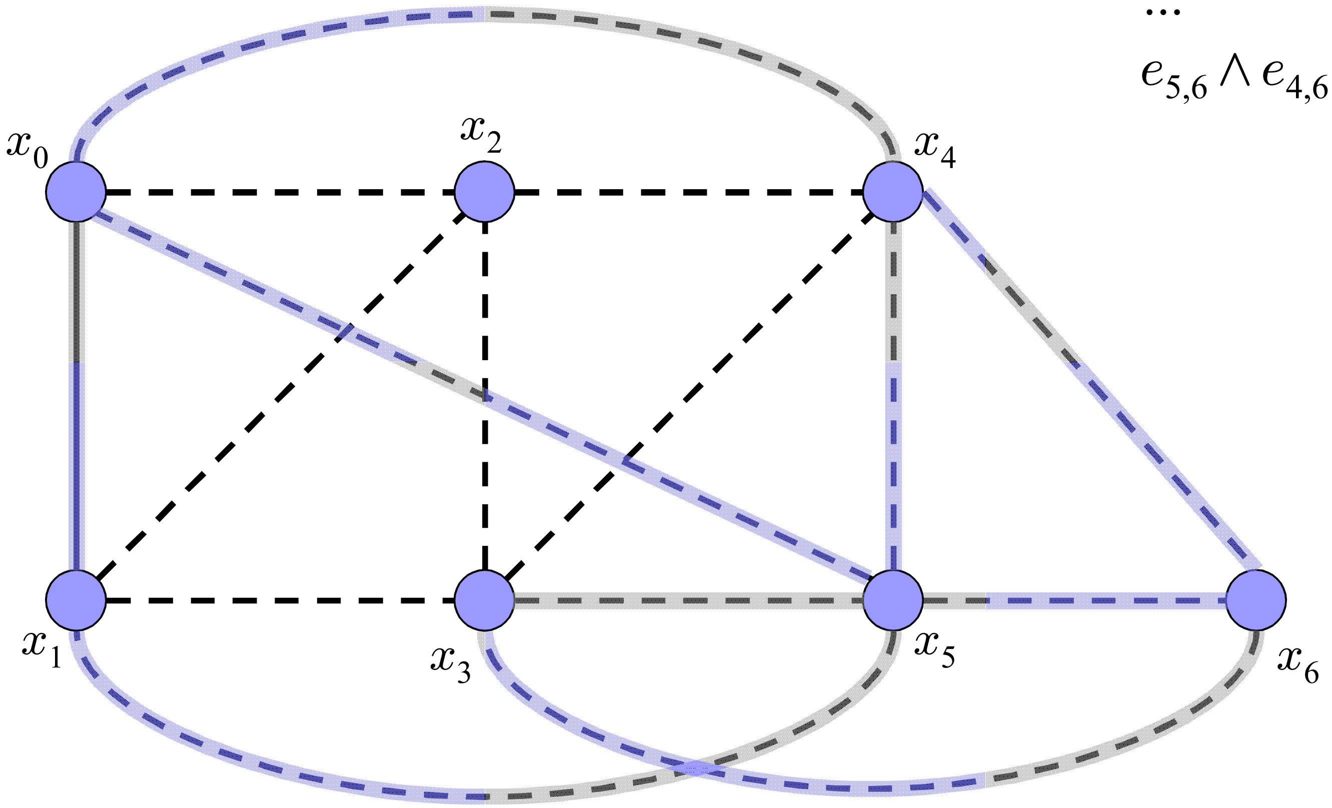
Suppose the graph has 3 more edges

Here we will stop, although …

cache:

…

$e_{5,6} \wedge e_{4,6} \rightarrow e_{4,5}$

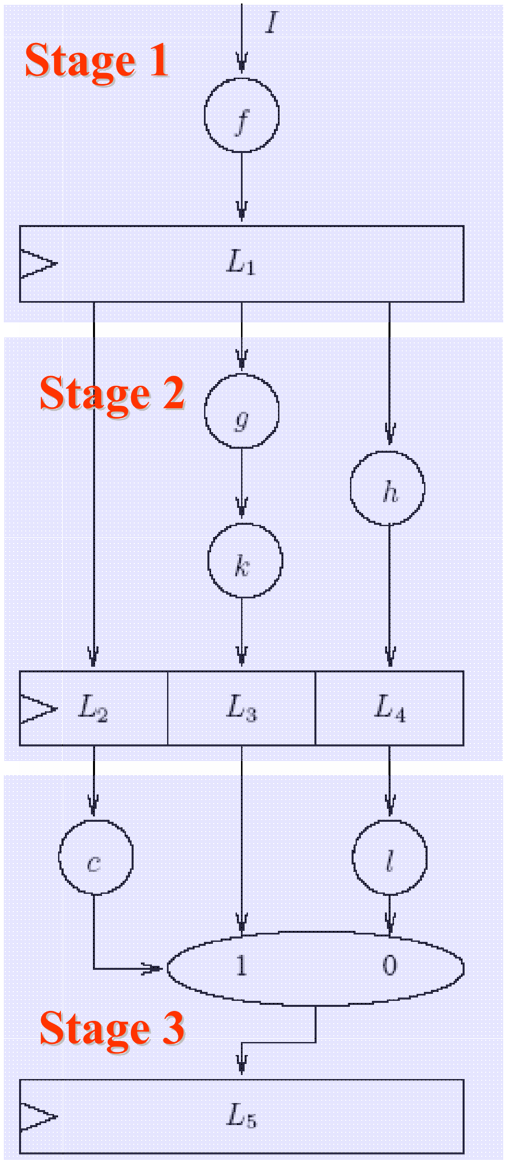


Open problem: constrain simple contradictory cycles in P time

Results

Benchmark set	# files	Sparse method				RTC			
		Constraints	Uclid	zchaff	total	Constraints	Uclid	zchaff	total
TV	9	16719	148.48	1.08	149.56	16083	151.1	0.96	152.06
Cache.inv	4	3669	47.28	40.78	88.06	3667	54.26	38.62	92.88
Dlx1c	3	7143	18.34	2.9	21.24	7143	20.04	2.73	22.77
Elf	3	4074	27.18	2.08	29.26	4074	28.81	1.83	30.64
ooo	6	7059	26.85	46.42	73.27	7059	29.78	45.08	74.86
Pipeline	1	6	0.06	37.29	37.35	6	0.08	36.91	36.99
Total	26	38670	268.19	130.55	398.74	38032	284.07	126.13	410.2
TV	9	103158	1467.76	5.43	1473.19	9946	1385.61	0.69	1386.3
Cache.inv	4	5970	48.06	42.39	90.45	5398	54.65	44.14	98.79
Dlx1c	3	46473	368.12	11.45	379.57	11445	350.48	8.88	359.36
Elf	5	43374	473.32	28.99	502.31	24033	467.95	28.18	496.13
ooo	6	20205	78.27	29.08	107.35	16068	79.5	24.35	103.85
Pipeline	1	96	0.17	46.57	46.74	24	0.18	46.64	46.82
q2	1	3531	30.32	46.33	76.65	855	32.19	35.57	67.76
Total	29	222807	2466.02	210.24	2676.26	67769	2370.56	188.45	2559.01

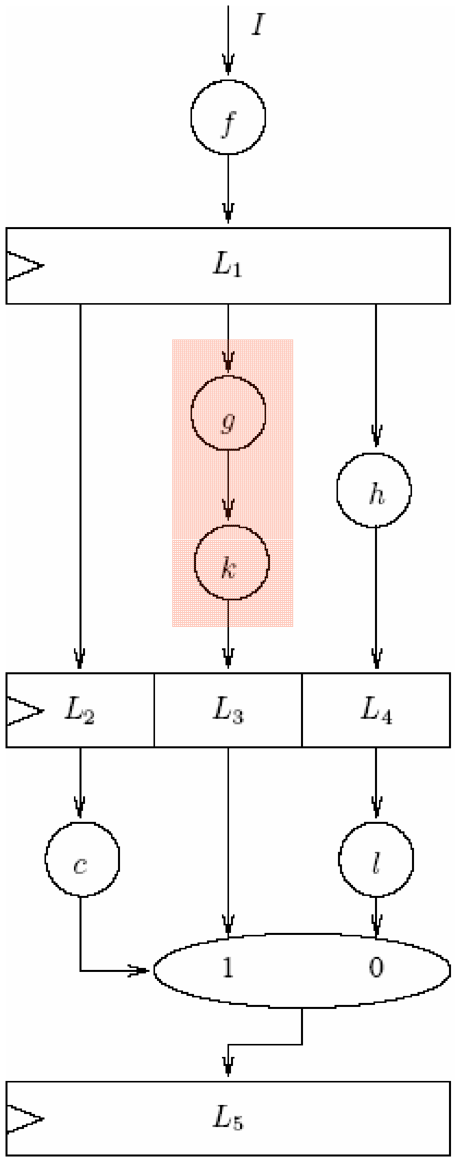
Example: Circuit Transformations



- A pipeline processes data in stages
- Data is processed in parallel – as in an assembly line
- Formal Model:

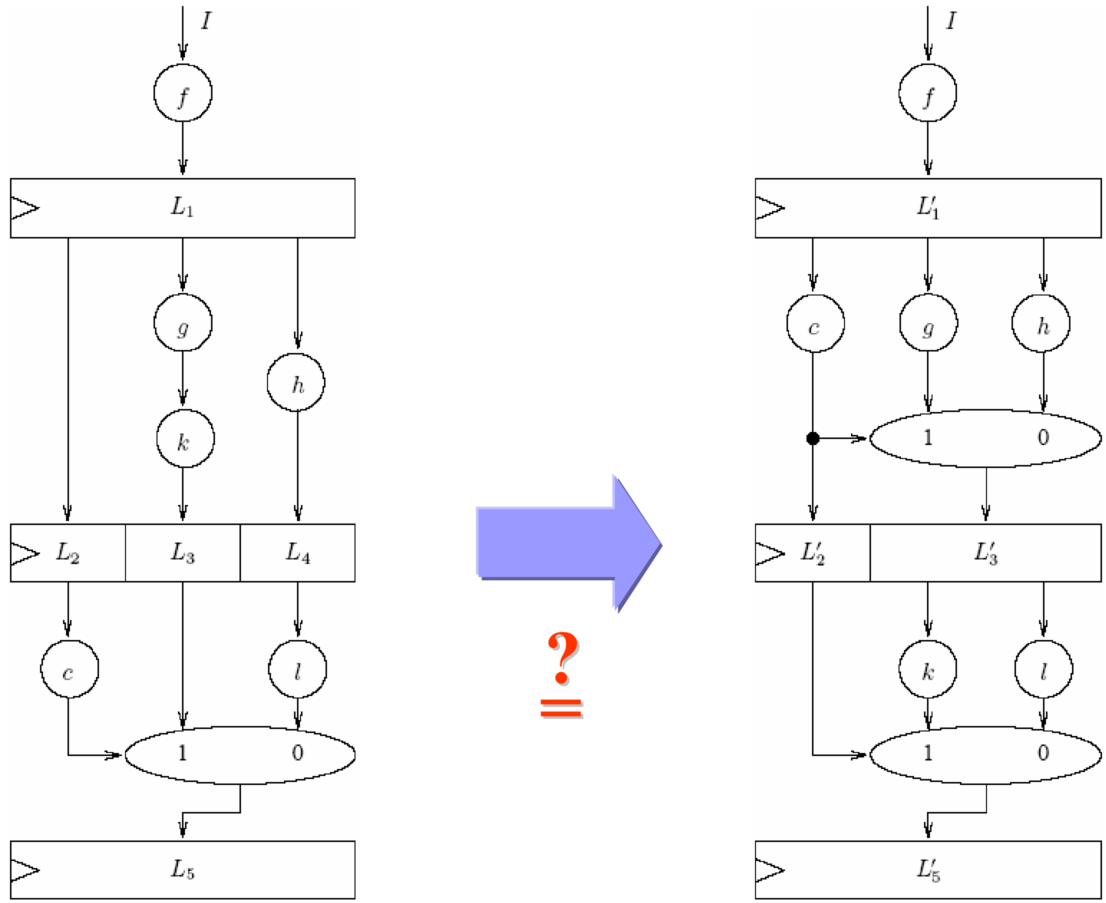
$$
\begin{aligned}
L_1 &= f(I) \\
L_2 &= L_1 \\
L_3 &= k(g(L_1)) \\
L_4 &= h(L_1) \\
L_5 &= c(L_2)\,?\,L_3 : l(L_4)
\end{aligned}
$$

Example: Circuit Transformations



- The maximum clock frequency depends on the longest path between two latches
- Note that the output of g is used as input to k
- We want to speed up the design by postponing k to the third stage

Validating Circuit Transformations



$$L_1 = f(I)$$
$$L_2 = L_1$$
$$L_3 = k(g(L_1))$$

$$L'_1 = f(I)$$
$$L'_2 = c(L'_1)$$
$$L'_3 = c(L'_1)\ ?\ g(L'_1$$

Validating a compilation process

- Target program

$$u_1 = x_1 + y_1;$$
$$u_2 = x_2 + y_2;$$
$$z = u_1 \cdot u_2;$$

Compilation

- Source program

$$z = (x_1 + y_1) \cdot (x_2 + y_2);$$

- Need to prove that:

$$(u_1 = \underbrace{x_1 + y_1 \wedge u_2 = x_2 + y_2 \wedge z = u_1 \cdot u_2}_{\text{Target}}) \leftrightarrow z = \underbrace{(x_1 + y_1) \cdot (x_2 + y_2)}_{\text{Source}}$$

Validating a compilation process

- Target program

$u_1 = x_1 + y_1;$

$u_2 = x_2 + y_2;$

$z = u_1 \cdot u_2;$

Compilation

- Source program

$z = (x_1 + y_1) \cdot (x_2 + y_2);$

- Need to prove that:

$$(u_1 = \underbrace{x_1 + y_1}_{f_1} \wedge u_2 = \underbrace{x_2 + y_2}_{f_2} \wedge z = \underbrace{u_1 \cdot u_2}_{g_1}) \leftrightarrow z = \underbrace{(\underbrace{x_1 + y_1}_{f_1}) \cdot (\underbrace{x_2 + y_2}_{f_2})}_{g_2}$$

Validating a compilation process

- Instead, prove:

$$\varphi^{UF} = (\ldots = f \ldots \wedge \ldots =$$

under functional consistency: for every uninterpreted function f

$$x = y \rightarrow f(x) = f(y)$$

- Need to prove that:

$$(u_1 = \underbrace{x_1 + y_1}_{f_1} \wedge u_2 = \underbrace{x_2 + y_2}_{f_2} \wedge z = \underbrace{u_1 \cdot u_2}_{g_1}) \leftrightarrow z = \underbrace{(x_1 + y_1)}_{f_1} \cdot \underbrace{(x_2 + y_2)}_{f_2}$$

- Which translates to (via Ackermann's reduction):

$$\varphi^{E} = \left((x_1 = x_2 \wedge y_1 = \right.$$