Logic Synthesis & Verification, Fall 2021
Programming Assignment 2:
OR Bi-Decomposition of Functions

Presenter: Yun-Rong Luo
Instructor: Jie-Hong Roland Jiang
ALCom Lab
EE Dept./ Grad. Inst. of Electronics Eng.
National Taiwan University

PA2 introduction

Overview

- Write a procedure in ABC that decides whether each circuit PO $f(X)$ is OR bi-decomposable.
- Integrate this procedure into ABC, so that after reading in a circuit by the command "**read**", running the command "**lsv_or_bidec**" would invoke your code.

Preliminaries

- A function $f(X)$ is OR bi-decomposable under a variable partition of its support $X = \{X_A | X_B | X_C\}$ if f can be written as $f(X) = f_A(X_A, X_C) \vee f_B(X_B, X_C)$
- A variable partition is **non-trivial** if $X_A \neq \emptyset$ and $X_B \neq \emptyset$. (In our PA, only non-trivial variable partition is of concern.)

Proposition 1

A function $f(X)$ can be written as $f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for some function f_A and f_B if and only if the Boolean formula $f(X_A, X_B, X_C) \wedge \neg f(X'_A, X_B, X_C) \wedge \neg f(X_A, X'_B, X_C)$ is unsatisfiable, where X'_A, X'_B are renamed versions of X_A, X_B, respectively.

- Proposition 1 assumes a given variable partition of its support $X = \{X_A | X_B | X_C\}$
- How to automate the process of finding a variable partition?

Automating finding variable partition

- For each $x_i \in X$, introduce two controlling variables α_i, β_i.
- $(\alpha_i, \beta_i) = (0,0), (0,1), (1,0), (1,1)$ indicates that $x_i \in X_C$, $x_i \in X_B$, $x_i \in X_A$, and x_i can be in either of X_A and X_B, respectively.

$$f(X) \land \neg f(X') \land \bigwedge_i \left((x_i \equiv x_i') \lor \alpha_i\right) \land \neg f(X'') \land \bigwedge_i \left((x_i \equiv x_i'') \lor \beta_i\right) \quad -(1)$$

- $\alpha_i = 0$ forces $x_i \equiv x_i'$, $\alpha_i = 1$ makes x_i' a free variable
  $\beta_i = 0$ forces $x_i \equiv x_i''$, $\beta_i = 1$ makes x_i'' a free variable

Incremental SAT solving

- Given a CNF ϕ, in **incremental SAT solving**, you can set **unit assumptions** $\vec{a}$ for variables (e.g. assuming $\alpha_i = 0, \beta_i = 1$). The assumptions can be thought as level-0 decisions. The SAT solver does SAT solving on $\phi \wedge \vec{a}$.
- If you want to do SAT solving for the same CNF ϕ but under a different set of assumptions $\vec{b}$, just free the previous assumptions $\vec{a}$ and set new assumptions $\vec{b}$. The SAT solver then solves $\phi \wedge \vec{b}$.
- The advantage of incremental SAT solving is: you can do SAT solving on the same CNF ϕ under different assumptions $\vec{b}$, without freeing the clauses of ϕ and the learnt clauses in previous assumptions $\vec{a}$.

- $(\alpha_i, \beta_i) = (0,0), (0,1), (1,0), (1,1)$ indicates that $x_i \in X_C$, $x_i \in X_B$, $x_i \in X_A$, and x_i can be in either of X_A and X_B, respectively.

Automating finding variable partition

$$f(X) \wedge \neg f(X') \wedge \bigwedge_i \left((x_i \equiv x_i') \vee \alpha_i\right) \wedge \neg f(X'') \wedge \bigwedge_i \left((x_i \equiv x_i'') \vee \beta_i\right) \quad -(1)$$

- A set of unit assumptions on controlling variables α_i, β_i that makes formula (1) unsatisfiable corresponds to a valid variable partition.
- E.g. Assume the support of f is $X = \{x_1, x_2, x_3\}$. If setting $(\alpha_1, \beta_1) = (0,0)$, $(\alpha_2, \beta_2) = (0,1)$, $(\alpha_3, \beta_3) = (1,0)$ makes formula (1) unsatisfiable, we obtain the variable partition $x_1 \in X_C, x_2 \in X_B, x_3 \in X_A$.
- But we don't want to enumerate all possible unit assumptions on controlling variables α_i, β_i! (Solution: **final conflict clause!**)

Final conflict clause

- Under a set of assumptions that makes the CNF unsatisfiable, SAT solver is able to provide a **final conflict clause** $C = (l_1 \vee l_2 \vee \cdots l_n)$ consisting of assumption literals.

```
veci        conf_final;    // If problem is unsatisfiable (possibly under assumptions),
                           // this vector represent the final conflict clause expressed in the assumptions.
```

- Setting assumptions that falsifies each l_i in C is sufficient to make the CNF unsatisfiable.
- E.g. The final conflict clause $(\alpha_1 \vee \beta_1 \vee \alpha_2 \vee \beta_3)$ indicates that setting $\alpha_1 = \beta_1 = \alpha_2 = \beta_3 = 0$ is sufficient for unsatisfiability, so setting $\beta_2 = \alpha_3 = 1$ doesn't affect the unsatisfiability, which suggests the variable partition variable partition $x_1 \in X_C, x_2 \in X_B, x_3 \in X_A$.

Avoiding trivial partition: seed partition

- Initially specify two distinct variable x_a, x_b and force $x_a \in X_A$, $x_b \in X_B$, and force $x_i \in X_C$, for $i \neq a, b$. That is, set the unit assumptions $(\alpha_a, \beta_a) = (1,0), (\alpha_b, \beta_b) = (0,1)$, and $(\alpha_i\ \beta_i) = (0,0)$, for $i \neq a, b$. This is called a **seed variable partition**.
- If the seed variable partition makes formula (1) unsatisfiable, the final conflict clause returned by SAT solver corresponds to a valid non-trivial variable partition.
- Every literal in the conflict clause is of positive phase because the conflict arises from a subset of the controlling variables $\alpha_i\ \beta_i$ set to 0.
- SAT solvers tend to return a conflict clause with few literals. The corresponding variable partition is desirable because $|X_C|$ tends to be small.

Avoiding trivial partition: seed partition

- If the seed partition fails, i.e. formula (1) is satisfiable under this seed partition, you should try another seed partition.
- For a given function $f(X)$ with $|X| = n$, the existence of non-trivial OR bi-decomposition can be checked with at most $(n-1) + \cdots + 1 = n(n-1)/2$ different seed partitions.

- $(\alpha_i, \beta_i) = (0,0), (0,1), (1,0), (1,1)$ indicates that $x_i \in X_C$, $x_i \in X_B$, $x_i \in X_A$, and x_i can be in either of X_A and X_B, respectively.

Output format

The format of your printing message is as follows.

```
PO <po1-name> support partition: 1
210200001101
PO <po2-name> support partition: 0
PO <po3-name> support partition: 1
212111000
```

Print lines of `PO <po-name>...` according to the order of `Abc_NtkForEachPo()`. For each PO, use `Abc_NtkCreateCone()` to extract the cone of the PO and its support set. In each line of `PO <po-name> support partition:`, print the names of POs returned by function `Abc_ObjName()`. Print `0` after `support partition:` if there is no valid non-trivial partition; print `1` if there is a valid non-trivial partition, and print the partition you find in the next line.

We use an integer string to represent the variable partition. Let `0` represents `x ∈ XC`, `1` represents `x ∈ XB`, `2` represents `x ∈ XA`.

For example, the string `212111000` indicates that the first support variable is in `XA`, the second in `XB`, the third in `XA`, . . . , the last in `XC`.

Programming tips

Programming tips

- See the wiki page on our GitHub: https://github.com/NTU-ALComLab/LSV-PA/wiki/Reasoning-with-SAT-solvers

What you need to do

- For each PO, use *Abc_NtkCreateCone()* to extract the cone of the PO and its support set.
- Use *Abc_NtkToDar()* to derives an equivalent *Aig_Man_t* from an *Abc_Ntk_t* network.
- Construct CNF of formula (1).
- Solve a non-trivial variable partition.

Constructing CNF

$$f(X) \wedge \neg f(X') \wedge \bigwedge_i \big((x_i \equiv x_i') \vee \alpha_i\big) \wedge \neg f(X'') \wedge \bigwedge_i \big((x_i \equiv x_i'') \vee \beta_i\big) \quad - (1)$$

- You can derive the CNF of $f(X)$ by *Cnf_Derive()* and write the clauses into a SAT solver object by *Cnf_DataWriteIntoSolver*.
- To obtain the CNF of $f(X')$ and $f(X'')$, use *Cnf_DataLift()* to obtain a copy of variable-renamed CNF of $f(X)$. You can write the new clauses into SAT solver by *sat_solver_addclause()*.
- You have to add clauses to assert the output phase of each copy $f(X), \neg f(X'), \neg f(X'')$.

Constructing CNF

- You have to add the clauses that represent $\bigwedge_i\left((x_i \equiv x_i') \vee \alpha_i\right)$ and $\bigwedge_i\left((x_i \equiv x_i'') \vee \beta_i\right)$. Note that $\left((x_i \equiv x_i') \vee \alpha_i\right)$ is not a CNF, you have to turn it into a equivalent form $\left((x_i \equiv x_i') \vee \alpha_i\right) = \left((x_i \Rightarrow x_i')(x_i' \Rightarrow x_i) \vee \alpha_i\right) = \left((\neg x_i \vee x_i')(\neg x_i' \vee x_i) \vee \alpha_i\right) = (\neg x_i \vee x_i' \vee \alpha_i)(\neg x_i' \vee x_i \vee \alpha_i)$.
- To do the last two steps, you need to memorize the variable number of $f(X), f(X'), f(X'')$ and each $x_i, x_i', x_i'', \alpha_i, \beta_i$ in CNF. You can just memorize the variable number for x_i and $f(X)$ (denoted as $varnum(x_i)$, $varnum(f(X))$ and the *VarShift* that you have to add to obtain the renamed version, e.g. $varnum(x_i) + VarShift = varnum(x_i')$.

Solving a non-trivial variable partition

- Iterate over all possible seed variable partitions, set unit assumptions on controlling variables α_i, β_i, and do incremental SAT solving until an unsatisfiable instance is found.
- You have to maintain an *assumpList* consisting of the unit assumptions of controlling variables α_i, β_i representing a seed variable partition. Unit assumptions are expressed as literals, α_i means setting $\alpha_i = 1$, $\neg\alpha_i$ means setting $\alpha_i = 0$.
- Pass the *assumpList* into *sat_solver_solve()*. Set the conflict limit for *sat_solver_solve()* to 0. That is, no limit is imposed.
- Extract the final conflict clause by *sat_solver_final()* and report the corresponding non-trivial variable partition.

How to know the usage of a piece of code?

- Trace the code of function definition.
- See how it is used in other part of ABC. You can run the following in the "src" directory, it recursively searches for the pattern *Abc_NtkCreateCone* under src.

```
grep -r "Abc_NtkCreateCone" ./
```

- "grep" reports

```
.//base/abci/abcDar.c:        pNtkOn1 = Abc_NtkCreateCone( pNtkOn, Abc_ObjFanin0(pObj), Abc_ObjName(pObj), 1 );
.//base/abci/abcDar.c:        pNtkOff1 = Abc_NtkCreateCone( pNtkOff, Abc_ObjFanin0(pObj), Abc_ObjName(pObj), 1 );
```

- You can see the example usage of *Abc_NtkCreateCone* in abcDar.c

```
Abc_NtkForEachCo( pNtkOn, pObj, i )
{
    pNtkOn1 = Abc_NtkCreateCone( pNtkOn, Abc_ObjFanin0(pObj), Abc_ObjName(pObj), 1 );
    if ( Abc_ObjFaninC0(pObj) )
        Abc_ObjXorFaninC( Abc_NtkPo(pNtkOn1, 0), 0 );
```

Reminder

- Submission period: 2021/12/24 11:00~13:00
- Please develop your code under **src/ext-lsv**. For those who develop your PA1 in **ext_<your_student_ID>**, please move your code to **src/ext-lsv** and remove **ext_<your_student_ID>**.
- Start and raise issues earlier!

Appendix

How to optimize the variable partition?

- A partition with smaller $|X_C|$ is more desirable.
- Given a final conflict clause $C = (l_1 \vee l_2 \vee \cdots l_n)$, which suggests that the assumption $l_1 = \cdots = l_n = 0$ is sufficient for unsatisfiability, you can try to set some $l_i = 1$ and see whether the CNF is still unsatisfiable.
- Try different seed partitions to find a more desirable partition.

How to derive f_A, f_B?

THEOREM 2 (CRAIG INTERPOLATION THEOREM). *[5] For any two Boolean formulas ϕ_A and ϕ_B with $\phi_A \wedge \phi_B$ unsatisfiable, then there exists a Boolean formula $\phi_{A'}$ referring only to the common input variables of ϕ_A and ϕ_B such that $\phi_A \Rightarrow \phi_{A'}$ and $\phi_{A'} \wedge \phi_B$ is unsatisfiable.*

- Interpolation is widely used in synthesis and verification, e.g. model checking, engineering change order (ECO), logic optimization and resynthesis.
- We will show how to derive f_A, f_B by interpolation.

Proof of Proposition 1

A function $f(X)$ can be written as $f_A(X_A, X_C) \vee f_B(X_B, X_C)$ for some function f_A and f_B if and only if the Boolean formula $f(X_A, X_B, X_C) \wedge \neg f(X_A', X_B, X_C) \wedge \neg f(X_A, X_B', X_C)$ is unsatisfiable, where X_A', X_B' are renamed versions of X_A, X_B, respectively.

($\Rightarrow$) If $f(X) = f_A(X_A, X_C) \vee f_B(X_B, X_C)$.
Then, $f(X_A, X_B, X_C) \wedge \neg f(X_A', X_B, X_C) \wedge \neg f(X_A, X_B', X_C)$
$= (f_A(X_A, X_C) \vee f_B(X_B, X_C)) \wedge \neg f_A(X_A', X_C) \wedge \neg f_B(X_B, X_C) \wedge$
$\neg f_A(X_A, X_C) \wedge \neg f_B(X_B', X_C) \equiv \perp$ is unsatisfiable.

Proof of Proposition 1

($\Leftarrow$) Construct $f_A(X_A, X_C), f_B(X_B, X_C)$ by interpolation.

- Assume $f(X_A, X_B, X_C) \wedge \neg f(X_A', X_B, X_C) \wedge \neg f(X_A, X_B', X_C) \equiv \perp$.
- Let $f(X_A, X_B, X_C) \wedge \neg f(X_A', X_B, X_C) = g(X_A, X_A', X_B, X_C)$.
- We have $g(X_A, X_A', X_B, X_C) \wedge \neg f(X_A, X_B', X_C) \equiv \perp$. By Theorem 2, we can generate some $f_A(X_A, X_C)$, where f_A only refers to common variables of $g(X_A, X_A', X_B, X_C)$ and $\neg f(X_A, X_B', X_C)$ and we have

$$g(X_A, X_A', X_B, X_C) \Longrightarrow f_A(X_A, X_C) \qquad -(2)$$

$$f_A(X_A, X_C) \wedge \neg f(X_A, X_B', X_C) \equiv \perp \qquad -(3)$$

THEOREM 2 (CRAIG INTERPOLATION THEOREM). *[5] For any two Boolean formulas ϕ_A and ϕ_B with $\phi_A \wedge \phi_B$ unsatisfiable, then there exists a Boolean formula $\phi_{A'}$ referring only to the common input variables of ϕ_A and ϕ_B such that $\phi_A \Rightarrow \phi_{A'}$ and $\phi_{A'} \wedge \phi_B$ is unsatisfiable.*

Proof of Proposition 1

- By Eq. (2), we also have $\neg f_A(X_A, X_C) \Longrightarrow \neg g(X_A, X_A', X_B, X_C)$ and

$$f(X_A, X_B, X_C) \wedge \neg f_A(X_A, X_C) \wedge \neg f(X_A', X_B, X_C)$$
$$\Longrightarrow f(X_A, X_B, X_C) \wedge \neg g(X_A, X_A', X_B, X_C) \wedge \neg f(X_A', X_B, X_C)$$
$$= g(X_A, X_A', X_B, X_C) \wedge \neg g(X_A, X_A', X_B, X_C) \equiv \perp.$$

- Let $f(X_A, X_B, X_C) \wedge \neg f_A(X_A, X_C) = h(X_A, X_B, X_C)$.
- We have $h(X_A, X_B, X_C) \wedge \neg f(X_A', X_B, X_C) \equiv \perp$. By Theorem 2, we can generate some $f_B(X_B, X_C)$, where f_B only refers to common variables of $h(X_A, X_B, X_C)$and $\neg f(X_A', X_B, X_C)$ and we have

$$h(X_A, X_B, X_C) \Longrightarrow f_B(X_B, X_C) \qquad -(4)$$
$$f_B(X_B, X_C) \wedge \neg f(X_A', X_B, X_C) \equiv \perp \qquad -(5)$$

Proof of Proposition 1

- Now, we claim that $f(X_A, X_B, X_C) \equiv f_A(X_A, X_C) \vee f_B(X_B, X_C)$. We prove the claim by showing bi-directional implications $f(X_A, X_B, X_C) \Leftrightarrow f_A(X_A, X_C) \vee f_B(X_B, X_C)$.
- The "$\Longrightarrow$" direction: By Eq. (4), we have $f(X_A, X_B, X_C) \wedge \neg f_A(X_A, X_C) \Longrightarrow f_B(X_B, X_C)$, which is equivalent to $f(X_A, X_B, X_C) \Longrightarrow f_A(X_A, X_C) \vee f_B(X_B, X_C)$.
- The "$\Leftarrow$" direction: By Eq. (3), (5), we have

$$f_A(X_A, X_C) \Longrightarrow f(X_A, X'_B, X_C) \quad -(6)$$
$$f_B(X_B, X_C) \Longrightarrow f(X'_A, X_B, X_C) \quad -(7)$$

Since f_A only refers to X_A, X_C, and X'_B is just a renamed version of X_B, we can rewrite Eq. (6), (7) into $f_A(X_A, X_C) \Longrightarrow f(X_A, X_B, X_C)$ and $f_B(X_B, X_C) \Longrightarrow f(X_A, X_B, X_C)$. Therefore, $f_A(X_A, X_C) \vee f_B(X_B, X_C) \Longrightarrow f(X_A, X_B, X_C)$.

Reference

- R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung. Bi-decomposing large Boolean func- tions via interpolation and satisfiability solving. In ACM/IEEE Design Automation Conference, pages 636–641, 2008. https://ieeexplore.ieee.org/document/4555896

THE END