Tesi

Andrea Locatelli

Contents

Introduzione

L'argomento principale di questa tesi è una particolare sotto categoria della crittografia, ovvero la Secure Multi-Party computation (SCM). Il protocollo di Secure Multi-party Computation si identifica come l'approccio maggiormente impiegato nella letteratura studiata per consentire a due o più parti di eseguire calcoli generici in modo collaborativo, mantenendo segreto il proprio input e condividendo solo il risultato finale.

Un particolare approfondimento svolto è stato quello relativo alla "Two party computation" e come renderla più efficiente. Si tratta di una particolare sotto-categoria delle SMPC che descrive uno scenario in cui due parti comunicano tra di loro per la risoluzione di un problema, senza però scambiarsi informazioni sensibili e senza ricorrere all'utilizzo di una terza parte fidata.

Una tecnica, largamente usata in questo ambito, si basa sulla sintesi di Garbled Circuits (GC) ed è stata proposta per la prima volta da Yao. Questo tipo di circuito, nelle sue varie fasi, permette di raggiungere l'obbiettivo di privacy sopra citato mediante l'utilizzo di un circuito virtuale in cui gli utenti inseriscono i propri input crittografati.

Per migliorare l'efficienza di questo tecnica e sfruttare i protocolli SMC in applicazioni pratiche, come l'esternalizzazione del calcolo in ambienti non attendibili, sono state introdotte numerose ottimizzazioni. In genere i circuiti utilizzati sono basati sulla logica booleana. Recentemente è stato proposto un protocollo di Secure Multi-party Computation basato invece sulle logiche multi valore. In questa tesi analizziamo, per la prima volta, l'implementazione di tecniche Multiple Valued Logic per la progettazione di Garbled Circuits, discutendo il loro impatto sui costi.

In particolare la finalità di questa tesi è quella di studiare se, applicando un circuito con logica multi valore, sia possibile ottenere una riduzione dei costi operazionali per conseguire una miglior performance del protocollo. Prima di intraprendere questa analisi è stato necessario soffermarsi su quali fossero i metodi più efficaci per ottenere una corretta conversione dei circuiti da binari a multi valore, che ci consenta di ottenere una buona minimizzazione dei costi totali e che allo stesso tempo ci renda possibile una comparazione dei risultati con i dati forniti dalla letteratura.

Un ulteriore strumento analizzato ed utilizzato è la sintesi dei circuiti, che viene applicata sia ai circuiti booleani che a quelli multi valore, in modo tale da ripulire la struttura da ridondanze e minimizzare il numero di porte il più possibile. Gli strumenti di sintesi in nostro possesso hanno giocato un ruolo fondamentale ai fini dell'analisi, in quanto i tool disponibili per gestire la logica multi valore non sono del tutto adatti ad affrontare questi tipi di nuovi problemi in quanto non implementati o aggiornati nel tempo. Per far fronte a questa carenza, la direzione della tesi si è spostata verso un approccio "ibrido".

L'obbiettivo del metodo proposto è quello di poter andare a utilizzare e valorizzare i punti di forza di entrambe le logiche. Nel caso della logica multi valore, riducendo il numero degli input che le parti devono utilizzare e limitando drasticamente la dimensione del dominio, è stato possibile ridurre i costi computazionali del processo. Dall'altra parte invece, nel caso della logica booleana, sfruttando gli strumenti di sintesi più evoluti ed implementati nel tempo è stato possibile ridurre notevolmente le dimensioni della struttura del circuito, riducendo ulteriormente i costi computazionali del processo.

I risultati finali della tesi hanno dimostrato che l'approccio ibrido può essere considerato interessante in quanto, nella maggior parte dei casi, è stato possibile notare una riduzione sostanziale dei costi del circuito in esame rispetto ad un normale circuito booleano sintetizzato

All'interno del primo capitolo di questa tesi viene introdotto il cambiamento a livello storico ed economico del valore dei dati degli utenti, che li ha resi un business primario per le aziende, portando alla monetizzazione di essi. Questo processo ha reso evidente una possibile problematica relativa alla violazione della privacy individuale della persona in quanto veniva attuato un massivo movimento di dati. Queste nuove dinamiche, le quali hanno definito le basi e le premesse dell'attuale processo per la sicurezza chiamato Secure multiparty computation (SMPC), hanno reso possibile operare sui dati senza esporli o spostarli dalla sede di partenza. In questo capitolo viene analizzato il funzionamento di questo processo atto ad elaborare metodi di calcolo congiunto di una funzione sugli input privati di due o più parti coinvolte proteggendo la privacy dei partecipanti l'uno dall'altro. Una volta definito il protocollo lo studio tratta quelle che vengono definite "garanzie di sicurezza" che rendono quest'ultimo efficace, i modelli di mondo reale/ideale al quale il paradigma viene associalo e i possibili avversari alla sicurezza che potrebbero partecipare a questo sistema.

Lo studio si focalizza in particolare su una sottocategoria dell'SMPC: la two party computation,
Essa descrive uno scenario in cui due parti parti comunicano tra di loro per la risoluzione di un problema, senza però scambiarsi informazioni sensibili e senza ricorrere all'utilizzo di una terza parte fidata. Questo scenario risulta analizzato per la prima volta tramite il Yao's garbled circuit protocol.

In seguito nel capitolo è esposta la struttura di questo tipo di circuiti, i quali risultano efficienti in quantità di passaggi e in sicurezza della struttura in quanto sono composti da circuiti booleani che permettono di "confondere" il circuito in modo che i partecipanti possano dedurre solo gli output senza venire a conoscenza di altri dati del sistema.

Il secondo capitolo si sviluppa partendo dall'utilizzo di un esempio per giungere ad una comprensione più approfondita del "Yao Garbled Circuit". Tramite questa dimostrazione è reso chiaro il metodo tramite il quale questo circuito è in grado di garantire un buon livello di sicurezza: i partecipanti modellano la funzione come un circuito booleano, e in seguito il mittente confonde i valori del circuito. Come conseguenza viene illustrato il protocollo chiamato oblivius tranfer, tramite il quale il mittente trasmette un pezzo di informazione al ricevitore, ma rimane ignaro di quale pezzo di informazione sia stato trasmesso, in questo modo il mittente non sarà in grado di conoscere l'input del ricevitore, conservando così la privacy dei dati.

In seguito all'esempio sopracitato, il capitolo necessita di un'ulteriore focus cruciale dedicato al garbling delle porte logiche e della loro tabella di verità. In questa sezione sono descritte le modalità con le quali è possibile confondere la tabella di verità e, alla fine del processo, criptare i valori di uscita adottando lo schema della crittografia simmetrica.

Una volta analizzate queste tematiche viene discussa la differenza tra un circuito booleano e un circuito confuso, in quanto questi due tipi di circuiti presentano delle differenze strutturali fondamentali riguardanti la semantica e la sintassi.

Il processo di confusione del circuito presenta un'ulteriore problematica: come avere la conferma del successo della decrittazione del circuito?. Per questo motivo viene analizzato di seguito l'espediente chiamato "permute and point" andandone a sviscerare la struttura e il funzionamento, per poi concludere il capitolo con un ulteriore esempio del permute and point in grado di mostrare effettivamente il funzionamento di questa tecnica.

Il terzo capitolo esordisce con una tematica molto importante dell'informatica in generale: l'efficienza del circuito. Applicando questa questione al nostro circuito confuso, questo capitolo è dedicato a due tool che vengono utilizzati come sintetizzatori, rispettivamente chiamati MVSIS e ABC. La funzione specifica di questi tool è quella di andare a ridurre quanto possibile le dimensioni dei circuiti dati loro in ingresso in modo da diminuire il costo totale dei circuiti eliminando nodi, ridondanze e inoltre cambiando e sintetizzando la struttura dei nodi. Nonostante entrambi abbiano lo stesso obbiettivo, essi lavorano utilizzando comandi e strumenti diversi.

Proseguendo nello sviluppo del capitolo si analizza per primo il tool MVSIS, descrivendone la struttura derivante dal tool SIS. MVSIS è stato implementato in modo tale da manipolare i circuiti con una logica detta "logica multi valore". Come verrà dettagliatamente descritto tramite esempi, esistono numerosi comandi in grado di svolgere funzioni sul circuito, tra queste quella di sintetizzare, di ottimizzare il circuito e molte altre per manipolare la struttura dei nodi. Oltre all'utilizzo dei sintetizzatori viene anche presentato un altro metodo che consiste nell'applicare metodi algebrici al fine di estrarre nuovi nodi che hanno divisori comuni per altri nodi con lo scopo di trovare sottoespressioni comuni, applicare una divisione semi-algebrica, attuare la decomposizione di una rete multi-valore e la fattorizzazione di una forma SOP.

In seguito, è presente l'analisi del tool ABC, che viene presentato come un successore di SIS, VIS e MVSIS., in quanto nato e creato con lo scopo di mantenere strutture di dati semplici e flessibili in modo da poter essere impiegate da una vasta gamma di applicazioni. Viene considerato un successore in quanto creato prendendo come base le esperienze di utilizzo dei tre tool sopracitati, i quali presentavano per ogni caso delle inadeguatezze differenti. Per meglio spiegare i funzionamenti di ABC, esso, nello svolgimento del capitolo, è stato messo a confronto con quello di MVSIS, evidenziandone le differenze ed esplicando quali sono i comandi utilizzabili nel sistema di ABC. Al termine del confronto viene spiegata la logica e i comandi che agendo all'unisono danno forma alla sintesi sequenziale.

Nel quarto capitolo si entra nel vivo della sperimentazione ed analisi convertendo in codice tutto ciò che è stato appreso dalla letteratura e dal lavoro di ricerca svolto nei capitoli precedenti. In particolare, nella prima parte del capitolo, l'attenzione è focalizzata sul testare l'efficacia dell'utilizzo di un multi valore rispetto all'utilizzo di un circuito booleano. Per poterli mettere a confronto viene presentata una ricerca sulle modalità esistenti finalizzate ad eseguire una conversione che però non vada ad intaccare la funzionalità del circuito. Questa conversione permetterà, tramite l'analisi sviluppata durante lo svolgersi del capitolo, di andare ad individuare quale dei due risulti essere più efficace e conveniente.

Il capitolo prosegue approfondendo le caratteristiche che competono la logica multi valore, andando a delineare quali sono la potenzialità che potrebbero risultare vantaggiose rispetto alla logica di un circuito booleano. In seguito a questa analisi è presentato un focus riguardante i circuiti binari. Il motivo di tale approfondimento risulta chiaro dal momento che, per testare la tesi, è stato necessario sviluppare una tecnica di conversione dei circuiti binari di partenza senza snaturarne la logica.

Una volta stabiliti i termini di questa conversione l'attenzione dello studio si sposta sulla ricerca di un nuovo dominio multi valore che fosse adatto all'analisi ma allo stesso tempo conveniente in termine di costi. Tutte queste operazioni svolte sui circuiti hanno portato con se la necessità di identificare un nuovo formato in grado di rappresentarli senza andarne a modificare la struttura. Il capitolo prosegue dunque con un'analisi approfondita di PLA (il formato selezionato per i circuiti binari), introducendo quindi l'elemento "don't care" che verrà sviluppato durante il proseguimento dello studio e applicato all'analisi svolta.

Oltre a PLA viene introdotto ed analizzato anche il formato blmf (sviluppato per i circuiti multi valore), necessario in quanto PLA non era sufficiente in quanto funzionale solo alla logica dei circuiti binari. Per implementare la conversione dei circuiti in analisi e renderli ready to use, viene inoltre utilizzato il linguaggio di programmazione Python.

Una volta approfonditi tutti questi argomenti il capitolo prosegue applicando la sintetizzazione dei circuiti risultanti mediante MVSIS E ABC, implementando e automatizzando i processi per poi valutare in fine i costi effettivi dei circuiti messi a confronto.

Il quinto capitolo di questa tesi contiene al suo interno i risultati sperimentali derivati dall'analisi svolta nei capitoli precedenti mirata a valutare se il nuovo approccio sviluppato potesse essere considerato una alternativa possibile o vincente rispetto alle proposte già presenti in letteratura.

Il capitolo mostra come la prima analisi sia stata svolta scegliendo l'approccio classico della logica binaria per provare ad applicarlo a quella multi valore, usando quindi MVSIS, al termine ritenuto non funzionale per due motivi: la mancanza di replicabilità di alcuni comandi in questo sistema e lo scarso sviluppo che il programma ha avuto nel tempo che lo rendono meno agile al fine dei calcoli e del suo utilizzo. L'analisi in questo senso ha mostrato come i costi vengono notevolmente abbassati nel caso del booleano mentre in quello multi valore rimangono più elevati. poichè la sintesi sui circuiti binari è nettamente più sviluppata e sono stati scoperti metodi di sintesi più efficaci rispetto a quelli studiati per una logica multi valore.

Proseguendo nelle analisi del capitolo è stato possibile appurare che, inizialmente il costo del circuito multi valore possiede dei costi più bassi per via del minor numero di input, ma che la logica binaria può appoggiarsi a degli strumenti di sintesi più sviluppati. Per questo motivo la sperimentazione procede provando ad applicare ABC.

La parte finale del capitolo parte del capitolo si focalizza dunque sull'utilizzo di questo tool. I circuiti vengono convertiti da multi valore a binari in modo da poter applicare gli algoritmi di sintesi di ABC i quali risultano più efficienti e sviluppati, generando però come output un circuito binario.

Il capitolo finale presenta una approfondita analisi derivata dai risultati sperimentali. Essi mettono in luce come l'utilizzo della logica multi valore all'interno dei circuiti potrebbe rivelarsi molto interessante in termini di riduzione dei costi, ma, nonostante questo, sia molto difficoltoso sviluppare un calcolo con MVSIS a causa dello sviluppo del tool troppo datato. Dall'altra parte invece il tool ABC, che risulta essere molto più sviluppato e all'avanguardia, accetta in ingresso circuiti multi valore, ma, per poter fare questo, nel suo motore di sintesi, essi vengono convertiti in circuiti booleani.

I calcoli effettuati hanno messo in luce come il miglior approccio applicabile sia quello ibrido, in quanto l'applicazione della logica multi valore dimezza i valori di imput e riduce drasticamente la dimensione del dominio, e, tramite ABC con la sua conversione interna booleana del circuito multi valore creato, i calcoli sono facilitati dalla continua evoluzione e dal continuo aggiornamento del tool e dei suoi algoritmi di sintesi.

Chapter 1

Secure multi-party computation

La Secure multi-party computation (SMPC) è una sottocategoria della crittografia che si pone come obiettivo quello di elaborare metodi di calcolo congiunto di una funzione sugli input privati di due o più parti coinvolte.

A differenza dei compiti crittografici tradizionali, secondo i quali la crittografia ha come fine ultimo quello di garantire la sicurezza e l'integrità della comunicazione o dell'archiviazione, e l'avversario è esterno al sistema dei partecipanti; la crittografia del modello informatico che andremo di seguito ad analizzare ha come finalità quella di proteggere la privacy dei partecipanti l'uno dall'altro.

Ci troviamo in un periodo storico in cui i dati, col passare degli anni, stanno diventando un elemento sempre più centrale per le dinamiche economiche ed informatiche e, in quanto tali, nuovo oggetto di interesse per le aziende. Il massivo incremento della loro importanza in campo economico e informatico trova una spiegazione nel cambiamento/ incremento della loro funzione. Se infatti inizialmente i dati venivano considerati come semplici "informazioni" relative agli utenti, attualmente essi si sono evoluti e trasformanti in potenziali nuovi metodi di profitto.

Questo processo di cambiamento della funzione e del valore dei dati ha portato nel tempo alla nascita di un nuovo modello di business primario basato prettamente sulla monetizzazione delle informazioni relative agli utenti. Questo modello di business è attualmente comune a molte delle più grandi aziende ed è diventato priorità di molte altre.

Per fare in modo che l'applicazione di un simile modello di business risulti efficace è necessario tuttavia effettuare la raccolta di grandi quantità di dati: ne consegue che il successo di tale operazione sarà tanto più significativo quante più correlazioni e rapporti di casualità sarà possibile trovare combinando diverse

fonti dati. E' necessario tuttavia considerare tutte le problematiche derivanti da questa nuova dinamica.

Uno dei sottoprodotti potenzialmente rischiosi di questo processo è, ad esempio, la possibile violazione della privacy individuale della persona in quanto le aziende, incrementando la quantità di dati, possiederanno una grande quantità di informazioni personali relative ai loro utenti.

Per far fronte a questa eventuale problematica sono state studiate e sviluppate molteplici soluzioni nelle quali sia possibile utilizzare fonti dati diverse, senza però dover necessariamente centralizzare l'informazione. Questa tecnica fa in modo che la privacy individuale degli utenti venga protetta e garantisce che nessuna informazione venga rivelata a terzi nel corso delle operazioni.

Le nuove dinamiche relative alla sicurezza dei dati stileranno dunque le basi e le premesse dell'attuale processo per la sicurezza chiamato Secure multiparty computation (SMPC). In altri termini, grazie al SMPC, sarà possibile consentire ai Data Scientist e Data Analyst di operare sui dati senza il bisogno di esporli o spostarli dalla loro sede di storage di partenza.[6]

1.1 Panoramica

In una SMPC prendiamo in considerazione un dato numero di partecipanti che, al fine delle analisi, chiameremo $p_1, p_2, ..., p_N$. Ad ognuno di essi verranno associati i propri dati privati che, invece, chiameremo $d_1, d_2, ..., d_N$. L'intento del processo nasce dalla necessità dei partecipanti di calcolare il valore di una funzione pubblica con i loro dati privati: $F(d_1, d_2, ..., d_N)$ mantenendo però i loro input segreti.

Facciamo un esempio: supponiamo di avere tre parti che chiameremo rispettivamente Alice, Bob e Charlie. A ciascuno di loro sono associati i rispettivi input x, y e z che denotano i loro stipendi. Essi hanno come obbiettivo quello di scoprire quale sia, tra i tre stipendi che percepiscono, quello più alto, senza però rivelare agli altri quanto guadagna ciascuno di loro.

Questa premessa si traduce Matematicamente nel seguente calcolo: $(x, y, z) = max(x, y, z)$

Se ci fosse una parte esterna fidata (ipotizziamo che i tre partecipanti abbiano un amico comune, lo chiameremo Tony, che deve necessariamente mantenere i segreti delle parti coinvolte), ognuno dei partecipanti potrebbe comunicare il proprio stipendio a Tony, così facendo lui potrebbe calcolare quale tra i tre è quello maggiore e, in seguito, riferire quel numero a tutti loro.

L'obiettivo di SMPC è quello di evitare la situazione in cui i partecipanti, una volta portato a termine il confronto, vengano a conoscenza di un ulteriore dato rispetto al proprio di partenza o rispetto all'output del processo. Per questo motivo il SMPC si pone come scopo quello di progettare un protocollo nel quale, scambiando messaggi solo tra di loro, Alice, Bob e Charlie possono ancora calcolare $F(x, y, z)$ senza però rivelare il loro stipendio agli altri e senza dover dipendere da terzi. Essi infatti, durante l'esecuzione di questo protocollo, non dovrebbero venire a conoscenza di più dati di quanti ne saprebbero interagendo con un ipotetico Tony incorruttibile e perfettamente degno di fiducia.

In particolare, tutto ciò di cui le parti possono venire a conoscenza è ciò che loro stesse possono apprendere dall'output del confronto e dal loro stesso input. Così, nell'esempio precedente, se l'output è z allora Charlie impara che il suo è il valore massimo, mentre Alice e Bob imparano (se x, y e z sono distinti), che il loro input non è uguale al massimo, e che il massimo è uguale a z. Lo scenario di base può essere facilmente generalizzato ai casi in cui le parti hanno diversi input, diversi output e la funzione fornisce valori diversi alle diverse parti.

Un protocollo di calcolo sicuro multipartitico deve offrire determinte garanzie di sicurezza, valide anche nei casi in cui alcune delle parti siano in collusione o cerchino di violarne le regole. Queste garanzie possono essere definite come qui in seguito:

- **Input Privacy**: Nessuna delle parti corrotte (o suo sottoinsieme) deve in nessun modo essere in grado di derivare alcuna informazione sui dati appartenenti alle altre parti, ad eccezione di quanto rivelato dal risultato finale dell'operazione.
- **Correctness**: Nessuna delle parti corrotte (o suo sottoinsieme) deve essere in grado di indurre una parte onesta a produrre un risultato errato. Un protocollo di calcolo sicuro multipartitico, infatti, deve offrire alcune garanzie di sicurezza, persino se alcune delle parti fossero in collusione o cercassero di violarne le regole.

1.2 Garanzie di sicurezza

Per fare in modo che un protocollo di calcolo a più parti sia considerato efficace, esso deve garantire degli standard di sicurezza e, nella crittografia moderna, la sicurezza di un protocollo è strettamente legata ad una dimostrazione della sicurezza stessa del protocollo. La dimostrazione di sicurezza è da considerarsi una dimostrazione matematica in cui l'affidabilità di un protocollo è ridotta a quella della sicurezza delle sue primitive sottostanti. Tuttavia non è sempre possibile formalizzare la verifica di sicurezza del protocollo crittografico basata sulla conoscenza delle parti e sulla correttezza del protocollo stesso.

Per i protocolli MPC, l'ambiente in cui il protocollo opera è associato alla definizione del paradigma del mondo reale e del mondo ideale. Non è possibile dire che le parti non imparino nulla, poiché devono imparare l'output dell'operazione e esso dipende direttamente dagli input. Inoltre la correttezza dell'output non è garantita poiché la correttezza dell'output dipende dagli input delle parti e si deve supporre che gli input siano corrotti. Il paradigma del mondo reale/mondo ideale afferma due mondi che possiamo cosi definire e descrivere:

- Nel modello del mondo ideale esiste una parte incorruttibile fidata a cui ogni partecipante al protocollo invia il suo input. Questa parte fidata calcola la funzione da sola e rimanda l'output appropriato ad ogni parte senza rivelare ulteriori dati se non l'output del sistema.
- Nel modello del mondo reale, al contrario, non esiste una parte fidata e tutto ciò che le parti possono fare è scambiare messaggi tra loro senza ricorrere ad un interlocutore esterno. Un protocollo è detto sicuro se non si può imparare di più sugli input privati di ogni parte nel mondo reale di quanto si potrebbe imparare nel mondo ideale. Nel mondo ideale nessun messaggio viene scambiato tra le parti, quindi i messaggi scambiati nel mondo reale non possono rivelare alcuna informazione segreta agli altri utenti.

Il paradigma mondo reale/mondo ideale fornisce una semplice astrazione delle complessità di MPC per consentire la costruzione di un'applicazione sotto la premessa che il protocollo MPC, nel suo nucleo, sia in realtà un'esecuzione ideale. Se l'applicazione è sicura nel caso ideale, allora è sicura anche quando viene eseguito un protocollo reale.

A differenza delle applicazioni crittografiche tradizionali, come la crittografia o la firma, si deve assumere che, in un protocollo MPC, l'avversario sia uno dei giocatori impegnati nel sistema (o che controlla le parti interne) e che dunque quella/e parti corrotte possano esse stesse colludere per violare la sicurezza del protocollo dall'interno.

Sia n il numero di parti nel protocollo e t il numero di parti che possono essere avversarie. I protocolli e le soluzioni per il caso di $t < n/2$ (cioè quando si assume una maggioranza onesta) sono diversi da quelli in cui non viene fatta tale assunzione, quelli cioè dove le parti avversarie all'interno del sistema siano presenti in numero maggiore rispetto alle parti oneste. Quest'ultimo scenario include l'importante caso di calcolo a due parti in cui uno dei partecipanti può essere corrotto e il caso generale in cui un numero illimitato di partecipanti è corrotto e collude per attaccare i partecipanti onesti.[2]

Gli avversari affrontati dai diversi protocolli possono essere classificati in base a quanto, nel loro tentativo di eludere il sistema, sono disposti a deviare dal protocollo. Ci sono essenzialmente due tipi di avversari, ognuno dei quali dà luogo a diverse forme di sicurezza (e ognuno dei quali si adatta a diversi scenari del mondo reale):

- **Sicurezza semi-onesta (passiva)**: In questo caso, si assume che le parti corrotte cooperino semplicemente per raccogliere informazioni dal protocollo, ma non si discostino dalle specifiche del protocollo. Questo è un tipo di avversario semplice e ingenuo, che produce quindi una sicurezza debole in situazioni reali. Tuttavia, i protocolli che raggiungono questo livello di sicurezza, impediscono la perdita involontaria di informazioni tra le parti (altrimenti collaboranti) e sono quindi utili se questa è l'unica preoccupazione. Inoltre, i protocolli nel modello semi-onesto sono abbastanza efficienti e sono spesso un primo passo importante per raggiungere livelli più alti di sicurezza.
- **Sicurezza malevola (attiva)**: In questo caso, l'avversario può arbitrariamente discostarsi dall'esecuzione del protocollo nel suo tentativo di ingannare. I protocolli che raggiungono la sicurezza in questo modello forniscono una garanzia di sicurezza molto alta. Il problema sorge Nel caso in cui si presenti una maggioranza delle parti che si comportano scorrettamente. In questo specifico caso, cioè quello di maggioranza disonesta, L'unica cosa che un avversario può fare è far sì che le parti oneste interrompano il processo avendo rilevato l'imbroglio. Ne consegue dunque che, se le parti oneste ottengono l'output, allora hanno la garanzia che esso sia corretto. Anche in questi casi La loro privacy è sempre preservata.

La sicurezza contro gli avversari attivi porta tipicamente ad una riduzione dell'efficienza che porta alla sicurezza nascosta, una forma ridotta di sicurezza attiva. La sicurezza nascosta cattura situazioni più realistiche, in cui gli avversari attivi sono disposti a barare ma solo nel caso in cui non vi è la possibilità che il loro tentativo di imbroglio possa venire scoperto. Questo accade poiché, per esempio, tale dinamica potrebbe compromettere o danneggiare la loro reputazione impedendogli così di poter collaborare in futuro con altre parti oneste.

Conseguenza di tutto ciò è che i protocolli che sono segretamente sicuri forniscono meccanismi per garantire che, se alcune delle parti non seguono le istruzioni, allora sarà notato con alta probabilità. In un certo senso gli avversari segreti sono quelli considerati attivi ma costretti ad agire passivamente a causa di preoccupazioni esterne non crittografiche (ad esempio la possibilità di compromettere il business). Questo meccanismo stabilisce un ponte tra i due modelli nella speranza di trovare protocolli che siano efficienti e abbastanza sicuri nella pratica.

Come per molti protocolli crittografici, la sicurezza di un protocollo MPC può basarsi su alcuni diversi presupposti:

- **Computazionale** (cioè basata su un determinato problema matematico, come la fattorizzazione) o **incondizionata**, cioè basata sull'indisponibilità fisica dei messaggi sui canali (di solito con qualche probabilità di errore che può essere resa arbitrariamente ridotta).
- **Il modello potrebbe assumere diverse dinamiche**: ad esempio che i partecipanti utilizzino una rete sincronizzata dove un messaggio inviato a un "tick" arriva sempre al "tick" successivo, oppure che esista un canale di trasmissione sicuro e affidabile, o ancora che esista un canale di comunicazione sicuro tra ogni coppia di partecipanti dove un avversario non può leggere, modificare o generare messaggi nel canale, ecc.

1.3 Two-party computation

Una particolare sotto-categoria delle SMPC che andremo di seguito ad analizzare è la Two-party computation. Questa categoria descrive uno scenario in cui due parti comunicano tra di loro per la risoluzione di un problema, senza però scambiarsi informazioni sensibili e senza ricorrere all'utilizzo di una terza parte fidata. Lo scenario a due parti è particolarmente interessante ai fini di un'analisi in quanto non solo presenta delle diversità dal punto di vista delle applicazioni, ma anche perché a questo scenario a due si possono applicare tecniche speciali che non si applicano nel caso a più parti. Infatti, il calcolo sicuro a più parti è stato presentato per la prima volta nell'impostazione a due parti. Il lavoro originale è spesso citato come proveniente da uno dei due articoli di Yao; anche se gli articoli non contengono effettivamente quello che ora è noto come il Yao's garbled circuit protocol.

Il protocollo di base di Yao è sicuro contro gli avversari semi-onesti. E' possibile ritenere questo protcollo estremamente efficiente in termini di numero di passaggi in quanto questi sono costanti e può essere definito indipendente dalla funzione obiettivo che viene valutata. La funzione è vista come un circuito booleano, con ingressi in binario di lunghezza fissa. Un circuito booleano è un insieme di porte collegate con tre diversi tipi di fili: fili di ingresso al circuito, fili di uscita al circuito e fili intermedi. Ogni porta riceve due fili d'ingresso e ha un singolo filo d'uscita che potrebbe essere fan-out (cioè essere passato a più porte al livello successivo). La semplice valutazione del circuito viene fatta valutando ogni porta a turno e assumendo che le porte siano state ordinate topologicamente. Il gate viene rappresentato come una tabella di verità tale per cui, per ogni possibile coppia di bit (quelli provenienti dal gate dei fili di ingresso), la tabella assegna un unico bit di uscita, che è il valore del filo di uscita del gate. I risultati della valutazione sono i bit ottenuti nei fili di uscita del circuito.[19]

Yao ha spiegato come confondere un circuito (nascondere la sua struttura) in modo che due parti, mittente e ricevitore, possano imparare l'uscita del circuito e nient'altro. Ad un alto livello il mittente prepara il circuito confuso e lo invia al ricevitore che, ignaro, valuta il circuito imparando le codifiche corrispondenti all'uscita sua e del mittente. Una volta compiuta questa operazione si limita a rimandare le codifiche del mittente, permettendo a quest'ultimo di calcolare la sua parte di output. Il mittente invia la mappatura dalle codifiche di uscita dei ricevitori e i bit al ricevitore, permettendo a quest'ultimo di ottenere la propria uscita. Nel capitolo successivo andremo ad analizzare più in profondità questo processo, spiegandone dettagliatamente la logica di funzionamento. [17]

Chapter 2

Yao Garbled Circuit

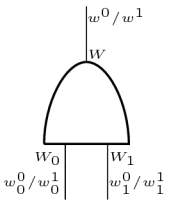
Per meglio comprendere il funzionamento del processo lo analizzeremo utilizzano un esempio:

Si supponga che Alice e Bob siano disposti a calcolare in modo sicuro una funzione mantenendo segreti i rispettivi input e .

Per fare ciò, essi modellano prima la funzione come un circuito booleano, questo è possibile poiché esiste un circuito booleano che calcola l'uscita di per qualsiasi funzione con ingressi di dimensione fissa. Tuttavia, il modo in cui tale modellazione viene eseguita può dipendere dalla funzione e non sarà ulteriormente discusso qui. Successivamente Alice confonderà il circuito booleano e:

1. Per ogni filo w_i del circuito C, sceglie casualmente due valori segreti w_i^0, w_i^1, dove w_i^j è il valore confuso del valore $j \in \{0, 1\}$ del filo w_i. Si noti che w_i^j non può rivelare j di per sé, quindi Alice deve tenere traccia di i e j. Questo deve essere fatto per ogni singolo filo di ingresso e di uscita di ogni porta logica del circuito, tranne che per le porte di uscita del circuito che possono essere lasciate in chiaro.
2. Alice dovrà costruire una tabella di verità confusa (GTT) T_i per ciascuna delle porte logiche G_i in C.

Queste tabelle devono essere tali che dati valori confusi lungo il suo insieme di fili d'ingresso, T_i permetterà di recuperare l'uscita confusa di questo G_i e nessun'altra informazione. Questa dinamica è possibile ottenerla attraverso la crittografia dei valori di uscita. Di seguito dettaglierò ulteriormente il garbling delle porte.



W_0	W_1	W
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.1: AND gate con etichette e tabella di verità.

In seguito, Alice può tradurre ogni bit del suo input nei suoi corrispondenti valori confusi sui fili di ingresso del circuito. Successivamente può inviare il circuito confuso a Bob con il suo input criptato.

Dopo che Bob ha ricevuto il circuito confuso, poiché tutti i fili d'ingresso sono criptati e solo Alice conosce la mappatura dei valori criptati e i bit reali, Bob ha bisogno di eseguire un Oblivious transfer con Alice per ciascuno dei suoi bit d'ingresso, in modo che Alice possa informarlo di quali valori criptati corrispondono ai suoi bit d'ingresso e sapere quali sono i suoi bit d'ingresso reali.

> **Oblivius transfer**: tipo di protocollo in cui il mittente trasmette un pezzo di informazione a un ricevitore, tra tante potenziali, ma rimane ignaro al mittente quale pezzo di informazione sia stato trasmesso.

Quindi significa che per ogni filo di ingresso, Bob sceglierà una tra le due stringhe casuali w_i^0, w_i^1 che corrispondono rispettivamente a 0 e 1, senza però conoscere il contenuto della stringa che non sceglie. Al fine di mantenere i dati in sicurezza, grazie alle proprietà del Oblivious transfer, Alice non potrà dunque conoscere l'input di Bob.

Arrivati a questo punto del processo Bob ha a sua disposizione tutti i valori necessari per calcolare l'uscita del circuito, come discuterò in seguito. Una volta fatto ciò, può comunicare i valori di uscita ad Alice. Seguendo questo procedimento Bob è stato in grado di ottenere l'uscita di senza rivelare il suo input, né conoscere l'input di Alice, questo significa che Alice e Bob hanno simulato con successo una terza parte fidata e hanno eseguito un SMPC sicuro in cui è garantita la privacy di entrambi gli utenti partecipanti al sistema.

2.1 Garbling Logical Gates

La nozione di garbling delle porte logiche e della loro tabella di verità risulta essere un punto cruciale in questa dinamica. Senza perdita di generalità, considererò solo le porte logiche con due fili di ingresso e un filo di uscita. Come spiegato precedentemente, per una data porta $G \in C$ e i suoi fili d'ingresso W_0, W_1 e il suo filo di uscita W, Alice doveva scegliere sei diverse stringhe casuali, $w_0^0, w_0^1, w_1^0, w_1^1, w^0, w^1$ che ha assegnato a ciascun valore dei fili in una mappatura uno a uno, dove w_i^j rappresenta la stringa casuale assegnata al valore j del filo W_i.

Quindi, per confondere la tabella di verità di G in modo da non rivelare alcuna informazione dati due valori di ingresso w_0, w_1 eccetto il suo valore di uscita w e nemmeno il tipo di porta logica, Alice può criptare i valori di uscita w_0, w_1 usando i valori di ingresso confusi come chiavi, adottando un dato schema di crittografia simmetrica $\mathbb{E}$.

Utilizzeremo la notazione $\mathbb{E}_{k_0,k_2}(x) = \mathbb{E}_{k_0}(\mathbb{E}_{k_1}(x))$ per indicare la cifratura doppia con due chiavi date k_0, k_1. Come esempio, criptiamo la tabella di verità della porta AND della figura:

W_0	W_1	W	Garbled value
w_0^0	w_1^0	w^0	$\mathbb{E}_{w_0^0,w_1^0}(w^0)$
w_0^0	w_1^1	w^0	$\mathbb{E}_{w_0^0,w_1^1}(w^0)$
w_0^1	w_1^0	w^0	$\mathbb{E}_{w_0^1,w_1^0}(w^0)$
w_0^1	w_1^1	w^1	$\mathbb{E}_{w_0^1,w_1^1}(w^1)$

La GTT T del gate G è semplicemente l'insieme $\left\{\mathbb{E}_{w_0^j,w_1^k}(w^{G(j,k)}) \mid j,k \in \{0,1\}\right\}$ dei valori confusi, dove $G(j,k)$ corrisponde all'uscita della porta G sotto ingressi (j,k).

2.2 Valutazione del Garbled Circuit

Una volta che Bob ha ricevuto il circuito confuso C da Alice e ha ottenuto i valori confusi del suo input attraverso diversi Oblivious transfer, egli può valutare il circuito.

Come premessa è però importante capire che un circuito confuso differisce da un normale circuito booleano per alcuni fattori: In un circuito booleano, infatti, semantica e sintassi sono fondamentalmente le stesse. Questo significa che vengono assegnati ad ogni filo due possibili valori semantici, cioè Vero o Falso, e che ,quest'ultimi ,sintatticamente si denotano come un segnale con valori 1 o 0 rispettivamente.

I segnali sopra citati sono considerabili pubblici e, questi stessi segnali sono associati ad ogni filo e chiunque può dedurre dal segnale quale sia il suo valore semantico. Questa condizione cambia però in un circuito confuso.

Nelle dinamiche di un circuito confuso, infatti, i valori semantici riferiti ad ogni segnale, eccetto quelli di uscita del circuito, sono mantenuti segreti e i segnali variano da un filo all'altro.

In questo modo, per valutare il circuito, per ogni portaG_i del circuito, Bob può provare a decifrare i valori nella tabella di verità associata T_i usando i valori di ingresso della porta come chiavi. Una delle voci in T_i sarà poi decifrata nell'uscita del gate.

questo punto del processo, sembrerebbe necessario avere un oracolo come unica strada praticabile per la conferma del successo della decrittazione delle voci di T_i; tuttavia è possibile ricorrere ad un "trucco" che descriverò più dettagliatamente qui di seguito.

Questo particolare espediente viene chiamato permute-and-point e viene usato per la prima volta [**?**] successivamente spiegato più nel dettaglio nella tesi di Phillip Rogoway[1].

Il permute-and-point permette di decidere quale voce della GTT deve essere decrittata dati gli input confusi, permettendo di eseguire calcoli più veloci e impedendo comunque al valutatore di dedurre qualcosa dall'ordine delle voci della tabella di verità. [10]

2.3 Permute-and-point

Il meccanismo di Permute-and-point funziona nel seguente modo: per ogni filo di ingresso e uscita w_i, Alice concatena un bit casuale $a \in \{0, 1\}$ alla fine del suo valore confuso w_i^0 e concatena il suo valore inverso $b = \overline{a} = 1 - a$ alla fine di w_i^1. Così facendo è dunque possibile associare ciascuna delle 4 permutazioni di 2 bit ad una delle entrate della GTT, senza però avere alcuna correlazione tra i bit e i valori della tabella della verità non confusa. In questo modo Alice può semplicemente ordinare la GTT secondo l'ordinamento naturale e darla a Bob che sarà quindi in grado di dedurre, senza bisogno di ulteriori informazioni, quale voce deve decifrare su un dato input. Per ottenere una rappresentazione ancora più chiara di questo "trucco", quei bit possono essere rappresentati come una coppia di colori, come è possibile osservare nella figura 2.2, in cui si notano le modalità con cui la tabella di verità viene modificata in modo da tenere conto di questo metodo.

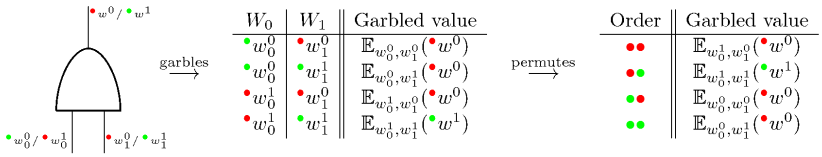


Figure 2.2: AND garbled con permute on point

Queste modifiche permettono a Bob di decifrare semplicemente la voce il cui indice corrisponde ai colori associati ai suoi fili di ingresso e quindi di ottenere il valore del filo di uscita e il suo colore, permettendogli di valutare ulteriormente il circuito. [20]

2.4 Esempio di Valutazione

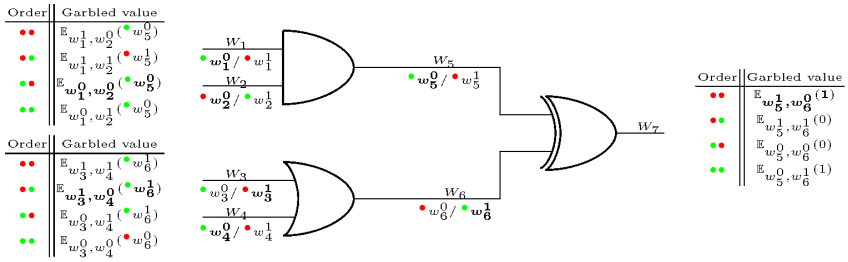


Figure 2.3: AND garbled con permute on point

Andiamo ora ad analizzare come si potrebbe valutare il circuito confuso rappresentato nella Figura 2.3 usando il metodo permute-and-point che abbiamo discusso precedentemente. Assumiamo che i valori semantici di ingresso di (W_1, W_2, W_3, W_4) siano $(0, 0, 1, 0)$, il che significa che l'input confuso effettivo è $(\bullet w_1^0, \bullet w_2^0, \bullet w_3^1, \bullet w_4^0)$ dove gli w_j^i sono i valori casuali che Alice ha scelto quando ha confuso il circuito, come visto sopra. Assumiamo anche che Alice abbia già fornito il suo input confuso, diciamo (W_1, W_3), e che Bob abbia già ottenuto il suo input confuso (W_2, W_4) da Alice attraverso due applicazioni di Oblivious transfer come descritto nel paragrafo riguardante.

Bob comincerà quindi a valutare prima la porta AND utilizzando l'ingresso $(\bullet w_1^0, \bullet w_2^0)$, dato che ha i colori $\bullet\bullet$ cercherà di decifrare la terza voce della GGT della porta AND, che funziona e quindi gli fornirà il valore confuso $\bullet w_5^0$.

Potrà poi continuare la sua valutazione con la seconda porta, che è la porta OR. Guardando il suo ingresso $(\bullet w_3^1, \bullet w_4^0)$ e proverà a decifrare la voce corrispondente a $\bullet\bullet$ con le chiavi (w_3^1, w_4^0), essa si decifra nel valore confuso $\bullet w_6^1$.

Potrà ora decifrare la porta XOR finale usando l'ingresso calcolato $(\bullet w_5^0, \bullet w_6^1)$, decifrando così la voce $\bullet\bullet$ che gli fornisce il risultato finale: 1.

Bob non è a conoscenza di quale sia stato effettivamente l'input di Alice, ne conosce solo l'output finale "1" e le stringhe generate casualmente $w_1^0, w_2^0, w_3^1, w_4^0, w_5^0, w_6^1$. Egli può ancora, per esempio, dedurre dal circuito che i valori semantici di w_5^0 e w_6^1 sono opposti, tuttavia questo sistema non gli permette di invertire il circuito confuso fino ai valori di ingresso di Alice.

Ci sono alcuni circuiti in cui non è assicurata la privacy, come per esempio un circuito che calcola la somma degli ingressi. Tuttavia, in questo caso di esempio Bob, conoscendo i suoi valori semantici di ingresso, può semplicemente limitare gli ingressi di Alice a un sottoinsieme dei possibili ingressi, ma non può determinare in modo univoco i reali valori di ingresso di Alice. [9]

Chapter 3

La sintesi dei circuiti

Nel capitolo precedente abbiamo discusso dell'importanza della presenza di un circuito logico all'interno del protocollo creato da Yao. A livello di funzionalità La scelta di un circuito efficiente è da considerarsi un fattore importantissimo in quanto rende la computazione e i tempi di calcolo delle operazioni non elevati. Ciò che effettivamente rende efficace la scelta è trovare il giusto bilanciamento tra numero di input che le 2 parti coinvolte devono immettere nel circuito e, nel caso di un dominio multi valore, scegliere un dominio non troppo elevato poiché quest'ultimo andrebbe ad immettere, per ogni valore di input, troppi valori che di conseguenza porterebbero ad aumentare i costi totali del circuito. [4]

La scelta di questi circuiti viene facilitata dall'utilizzo di strumenti chiamati sintetizzatori, questi tool sono stati creati appositamente per andare a ridurre quanto possibile le dimensioni dei circuiti dati loro in ingresso. All'interno di questo lavoro di analisi sono stati utilizzati due tool differenti: **MVSIS** e **ABC**. Entrambi sono stati sviluppati nel tempo dall'università di Berkley. Sia **MVSIS** che **ABC** contengono diversi metodi di sintesi capaci di andare ad eliminare i nodi e le ridondanze superflue all'interno dei circuiti e in grado di sintetizzare il circuito andandone a cambiare radicalmente la struttura interna dei nodi.

Nonostante l'uguale utilizzo per il quale questi due diversi tool sono stati sviluppati, vi sono comunque alcune differenze tra loro che andremo ad analizzare qui di seguito.

3.1 MVSIS

MVSIS è il primo tool di sintesi dei circuiti di cui parleremo. Grazie alla sua implementazione esso è in grado di manipolare i circuiti con una logica detta "logica multi valore". Per la sua creazione è stato preso come modello **SIS**, un tool sviluppato e specifico per la logica binaria, cercando di mantenerne simile la logica di funzionamento.

3.1.1 Design specification

Un circuito multi valore (MV circuit) può essere dato come valore di input a MVSIS tramite l'apposito comando (read_blifmv) e, oltre ai circuiti multi valore, è possibile anche dare come imput circuiti binari. Internamente, la rappresentazione del progetto è una rete di nodi MV dove ogni nodo rappresenta una funzione MV con una singola uscita a più valori. È importante però specificare una fondamentale distinzione con alcuni altri metodi multi valore, cioè che ogni variabile del nodo può avere un dominio diverso. L'intervallo per la variabile y_k è rappresentato dall'insieme $P_k={0,1,...,p_k -1 }. La funzione il cui esito dà UN SET dei mintermi (Una sola uscita con 0 o 1) per i quali $f_k = i$ (la funzione al nodo k è uguale al valore i) è chiamata *i-set* della funzione f_k e viene memorizzata in forma SOP (Somma di prodotti). Nel caso di logica a 2 valori, l'insieme 0 corrisponde all'off-set e l'insieme 1 all'on set. Una variabile MV yk è associata all'uscita del nodo k. Un margine congiunge k a j se uno qualsiasi degli i-set di j dipende esplicitamente da y_k. La rete ha un insieme di ingressi primari (che possono essere tutti a più valori) e un insieme di nodi di ingressi primari.

3.1.2 Semplificazione dei nodi

L'i-set (uno per ogni valore di output) di un nodo multi valore può essere semplificato con 2 comandi:

- SIMPLIFY
- FULLSIMP

FULLSIMP, tra le 2 opzioni, è quella che garantisce una semplificazione più efficace poichè Utilizza il CODC (Compatible Observability Don't Cares) e l' MV-image computation. Ogni i-set del nodo è in seguito semplificato dal metodo di sintesi ESPRESSO-MV utilizzando i valori che siamo stati in grado di ricavare ricorrendo alle due tecniche precedenti. [13]

3.1.3 Kernel e Cube Extraction

Oltre all'utilizzo dei sintetizzatori vi è però un altro importante step che è possibile applicare per operare un' ottimizzazione del circuito: questo ulteriore passaggio consiste nell'applicare metodi algebrici al fine di estrarre nuovi nodi che hanno divisori comuni per altri nodi. Nel tempo sono state sviluppate nuove tecniche algebriche dedicate alla logica MV che ci permettono di trattare uniformemente le variabili binarie e multi-valore. Tra le loro funzioni queste tecniche algebriche includono svariate possibilità, tra cui: metodi per trovare sottoespressioni comuni, divisione semi-algebrica, decomposizione di una rete multi-valore e fattorizzazione di una forma SOP. Per fare maggiore chiarezza di seguito verranno elencati i comandi più rilevanti, seguiti da alcune brevi descrizioni delle loro capacità.

1. **FX**: questo comando individua divisori comuni all'interno della rete e crea nuovi nodi all'interno di essa, nei quali vengono raccolti tutti i nodi coinvolti nel divisore comune.
2. **Decomp**: questo comando esegue una completa fattorizzazione multivaloriale degli i-sets di ogni nodo e, in seguito, suddivide i nodi secondo queste fattorizzazioni. Dopo questo procedimento viene utilizzato il comando *resub*, che ricorre alla divisione algebrica multi valore per eliminare i fattori duplicati all'interno della rete.

3.1.4 Altri comandi per la manipolazione dei nodi

1. **Collapsing**: converte l'intera rete multilivello in modo che le forme SOP per ogni uscita siano in termini di soli ingressi primari. Così il numero di nodi nella rete sarà esattamente il numero di uscite primarie.
2. **Merging**: Acquisisce tutti i nodi della rete e forza un'unione di questi, così facendo genera un singolo nodo multi-valore costruendo un i-set per ogni combinazione di valori creata. Nel caso in cui da questo processo vengano generati i-set uguali, essi vengono uniti in un singolo nodo.
3. **Encoding**: è considerabile come l'inverso del merging di funzioni binarie, esso infatti cerca di trovare una buona codifica binaria per ogni variabile multi-valore nella rete, compresi gli ingressi e le uscite primarie. Alla fine di questo procedimento ogni segnale è codificato come un segnale binario. È possibile quindi, grazie al comando encoding, scrivere un file binario a partire da un multi valore. Questo processo di traslazione può essere così spiegato in due fasi:

    1. Inizia dagli ingressi e per ogni nodo, determina se uno dei suoi fanin può essere usato per codificare parzialmente il nodo.
    2. nizia dalle uscite e, in ordine topologico inverso, lavora a ritroso fino agli ingressi primari. Ad ogni nodo, le sue uscite sono codificate utilizzando le informazioni su come sono utilizzati i suoi fanout.

4. **Pair decoding**: Ha la stessa funzione del merging, ma, a differenza del precedente, questo comando utilizza un altro metodo per scegliere quali nodi unire.
5. **Bi-decomposition**: Produce dei nodi multi-valore intermedi. Esso acquisisce una rete MV appiattita, o parzialmente appiattita, e ne genera una nuova composta da porte MAX e MIN multi-valutate a due ingressi e da iterali multi-valutati. In questo processo, per la creazione dei nuovi nodi, vengono sfruttate sia l'incompletezza della specifica iniziale che le flessibilità generate nel processo di composizione. [5] [18]

3.2 ABC

ABC è un sistema software in sviluppo che ha come obbiettivi la sintesi e la verifica di circuiti logici sequenziali binari che appaiono in progetti hardware. Il meccanismo di ABC si basa sulla combinazione di un'ottimizzazione logica scalabile basata su And-Inverter Graphs (AIGs), una mappatura tecnologica basata su DAG per tabelle di look-up, celle standard e algoritmi innovativi per la sintesi e la verifica sequenziale.

Si può dire che questo programma sia nato dalle esperienze assimilate dall'uso di SIS, VIS e MVSIS. Lo sviluppo di ABC è stato portato avanti in quanto questi sistemi precedentemente citati non erano in grado di fornire un ambiente di programmazione flessibile per implementare le recenti innovazioni. Per quanto riguarda l'ambiente SIS, esso risulta essere obsoleto e piuttosto inefficiente nella gestione di grandi circuiti. VIS, invece, essendo stato progettato come strumento di verifica formale per specifiche multi-valore, non fornisce abbastanza flessibilità per la sintesi binaria. Diverso è invece il discorso per quanto riguarda MVSIS che, nonostante il suo ampio utilizzo, presenta alcune problematiche:

- Le strutture dati e gli algoritmi di base di MVSIS possono essere resi considerevolmente più semplici e facili da utilizzare assumendo come punto di partenza le reti binarie.
- Un posto centrale nel nuovo sistema dovrebbe essere dato a una nuova struttura dati, le AIG (reti logiche multilivello composte da AND e inverter a due ingressi), che promettono miglioramenti nella qualità e nel tempo di esecuzione della sintesi e della verifica.

La comprensione di queste problematiche ha spinto a ri-sviluppare i pacchetti di base di MVSIS creando un nuovo ambiente di programmazione chiamato ABC. Come suggerisce il nome, l'obiettivo primario di questo sistema software è quello di mantenere strutture di dati semplici e flessibili in modo da poter essere impiegate da una vasta gamma di applicazioni.

L'obiettivo del progetto ABC è quello di fornire un'implementazione pubblica degli algoritmi di sintesi combinatoria e sequenziale allo stato dell'arte e, allo stesso tempo, creare un ambiente open-source, in cui tali applicazioni possano essere sviluppate e confrontate. La versione attuale di ABC può ottimizzare, mappare e realizzare progetti industriali a livello di gate con 100K porte e 10K elementi sequenziali cont tempi di calcolo non superiori al minuto prendendo in considerazione un computer moderno. [14]

3.2.1 Sintesi combinatoria

Per comprendere a pieno l'innovazione di ABC ci basta eseguire una comparazione con MVSIS. Infatti, i metodi di logica combinatoria utilizzati da ABC all'interno dei suoi script (resyn e resyn2) sono tipicamente 10-100 volte più veloci se messi a confronto con gli script utilizzati all'interno di SIS o MVSIS.

In ABC, i confini dei nodi sono inizialmente distrutti dall'hashing strutturale (comando strash), che trasforma una rete logica in un AIG. I confini possono essere ricreati su richiesta utilizzando il comando renode, che può essere considerato come un inverso del comando SIS eliminate. Nel flusso di sintesi presentato sopra la logica viene trasformata a livello di AIG senza creare nodi.

- **Balance**: prende come ingresso un AIG e lo bilancia in modo tale che la struttura risulti equilibrata per il calcolo.
- **Cleanup**: Mette in atto la rimozione dei nodi logici che non sono a ventaglio in PO e latches.
- **Collapse**: Collassa l'intero circuito ad una rete ad un solo livello. Le funzioni dei nodi sono rappresentate utilizzando le BDD.
- **Dsd**: Applica al circuito la decomposizione disjoint-support utilizzando l'algoritmo di Bertacco/Damiani [3].
- **fx**: Rileva la logica di condivisione estraendo i divisori a due cubi e i divisori a due lettere a un cubo sfruttando l'algoritmo di Rajski/Vasudevamurthi [16].
- **Multi**: Espande l'AIG a due ingressi generando una rete di porte AND a più ingressi.
- **refactor**: Esegue il collasso iterativo e il refactoring dei coni logici nell'AIG con lo scopo di ridurre sia il numero di nodi AIG che il numero di livelli logici.

- **renode**: Assume che l'input sia un AIG. Crea i confini dei nodi in quest'ultimo e collassa la logica intermedia per formare nodi più grandi.
- **rewrite**: Assume che l'input sia un AIG. Crea i legami tra i nodi in quest'ultimo e collassa la logica intermedia allo scopo di formare nodi più grandi.
- **rr**: Esegue la rimozione della ridondanza per le AIG.
- **strash**: Trasforma la rete data in imput al programma in un AIG tramite un livello di hashing strutturale. L'AIG risultante è una rete logica composta da due fattori: porte AND a due ingressi e invertitori rappresentati come attributi complementari sui bordi. L'hashing strutturale è una trasformazione puramente combinatoria che, in quanto tale, non modifica in alcun modo il numero e le posizioni delle porte.
- **sweep**: Grazie al suo funzionamento Sweep esegue i seguenti compiti: rimuove i nodi dangling (nodi senza fanout), collassa i buffer e gli inverter nei loro fanout, propaga le costanti, e rimuove i fanin duplicati. Sweep non può però essere applicato a un AIG perché quest'ultimo è strutturalmente hashed e quindi non ha buffer, invertitori e nodi costanti non propagati. Per rimuovere i nodi dandling nella rete logica è necessario dunque utilizzare cleanup. tramite questi comandi sweep è in gradi di ridisegnare completamente il circuito a livello strutturale.

3.2.2 Sintesi sequenziale

La sintesi sequenziale è un processo che, ove presenti, trasforma la rete corrente modificandone la logica insieme agli elementi di memorizzazione (latches o flip-flops). La rete risultante dalla sintesi sequenziale può avere una codifica di stato e uno spazio di stato diversi rispetto alla rete originale, ma le due reti sono sequenzialmente equivalenti; nello specifico, (partendo dagli stati iniziali, per le stesse sequenze di vettori d'ingresso vengono generate sequenze identiche di vettori d'uscita). [12]

- **Cycle**: Simula la rete sequenziale con un input casuale per poi aggiornare il suo stato attuale..
- **init**: Ripristina gli stati iniziali di tutte le porte della rete corrente.
- **lcorr**: Attua una implementazione suddivisa di registro-corrispondenza usando l'induzione semplice, per poi rilevare e fondere registri che siano sequenzialmente equivalenti.
- **retime**: Implementa diversi tipi di retiming, tra questi citiamo i seguenti:
  - most forward;
  - most backward;
  - minimum-register;
  - minimum-delay euristico;
  - delay-optimal retiming [15]

  Quando il circuito è trasformato dall'AIG in una rete logica, le porte sono condivise in modo ottimale attraverso gli archi di fanout. Il calcolo degli stati iniziali dopo il retiming è ridotto ad un problema SAT che viene risolto ricorrendo a MiniSat.
- **scleanup**: Esegue la pulizia sequenziale, cioè rimuove i nodi e le latches che non sono a fanout.
- **ssw**: Implementa i segnali corrispondenti applicando l'induzione K-step. In seguito Rileva e fonde i nodi sequenzialmente equivalenti.
- **undc**: Converte i registri con gli stati iniziali don't care in registri con uno stato iniziale costante -0. In particolare è possibile utilizzare questo comando prima di eseguire qualsiasi verifica sequenziale.
- **xsim**: Esegue la simulazione con X valore della rete sequenziale corrente.

Chapter 4

Analisi

Il protocollo di Yao prevede, durante i suoi scambi, che le due parti coinvolte concordino anche sulla trasmissione di un circuito a cui entrambi gli attori debbano inserire dei valori di input. Uno degli scopi della nostra tesi consiste nel testare l'efficacia dell'utilizzo di un multi valore rispetto all'utilizzo "classico" di un circuito booleano, concentrandoci specificatamente sull'importante valore costituito dal fattore di costo computazionale delle operazioni che vengono svolte.

Per avviare la nostra analisi siamo partiti quindi selezionando un set di circuiti binari con caratteristiche diverse, tra cui differente numero di ingressi, uscite e livelli. Partendo da questa base ci siamo successivamente concentrati sulla ricerca di una metodologia di conversione dei circuiti atta a traslarli da un 'classico' dominio booleano ad uno multi valore, dove quindi i valori non sono più rappresentati da 0 e 1 ma da un set più ampio di valori. Tutto ciò sempre mantenendo le funzionalità del circuito. Una volta ottenuta una controparte per ogni circuito si è proceduto con un confronto dei circuiti al fine di riuscire ad individuare quali di questi set fossero più efficaci.

4.1 La logica multi-valore

La logica multi-valore è una generalizzazione della classica logica booleana. Una delle ragioni per la quale è stata presa in considerazione è che essa può portare ad una comprensione più profonda di problemi specifici. La logica multi valore può avere numerosi vantaggi teorici rispetto a quelle booleana:

- Memory design: Memorizzare informazioni a blocchi di 2bit invece che uno raddoppia la densità delle informazioni che si possono racchiudere nello stesso spazio.
- Utilizzo per rappresentazione di funzioni booleane con più output;
- Design di PLA con input decoder;
- Ottimizzazione di macchine a stato finite;
- Testing e verifica.

Le potenzialità pratiche di questi vantaggi dipendono pesantemente dalla disponibilità della realizzazione di questi circuiti che devono comunque rimanere compatibili con gli standard tecnologici odierni. [8]

4.1.1 Le funzioni multi-valore

Una funzione multi-valore è una funzione discreta che ha, sia come input che come output, delle variabili che assumono più di 2 valori. Solitamente le funzioni multi-valore sono definite come: $f : P_1 \times P_2 \times ...P_n \rightarrow Q$ dove le variabili x_i della funzione prendono valore dal gruppo $P_i = \{0, 1, 2, ..., p_i - 1\}, p_i > 1, i \in \{1, 2, ..., n\}$ e la funzione prende valori dal set $Q = \{0, 1, 2, ..., q\}, q > 1$

4.1.2 Rappresentazione di funzioni multi-valore

Il modo più semplice per rappresentare un funzione con m valori e n variabili consiste nel creare una tabella di verità con m^n righe, ognuna delle quali specifica il valore della funzione corrispondente alla combinazione delle variabili in ingresso. Una tabella di verità può essere ridisegnata come un rettangolo n-dimensionale con m-valore in una mappa di Karneaugh. Il problema di queste due rappresentazioni consiste nella difficoltà di raffigurazione in caso di tabelle complesse. Nei prossimi capitoli mostrerò metodi di rappresentazione più efficaci per la logica multi valore.

4.1.2.1 Positional cube notation

Utilizzata nella logica booleana, questa tecnica di visualizzazione può essere applicata anche per funzioni con input multi valore e output booleano. A differenza di un letterale binario, x o x', viene rappresentato da un campo a 2 bit, una funzione letterale multi valore m viene rappresenta con un campo a m bit. Il valore $i \in M$ corrisponde al bit i dell campo. Un letterale x^S presenta il valore 1 in tutte le posizioni $i \in S$. Per esempio:

x1/x2	0	1	2
0	1	0	0
1	1	1	1
2	0	1	1

Applicando la Positional Cube Notation esso diventa:

$$\begin{array}{lcl} 110 & - & 100 \\ 101 & - & 010 \\ 011 & - & 001 \end{array}$$

Per quanto riguarda il caso booleano la condizione don't care di un valore in una riga corrisponde al fatto che quell'elemento può assumere valori 0 o 1. Per una funzione multi valore, invece, una condizione don't care può essere rappresentata nel seguente modo:

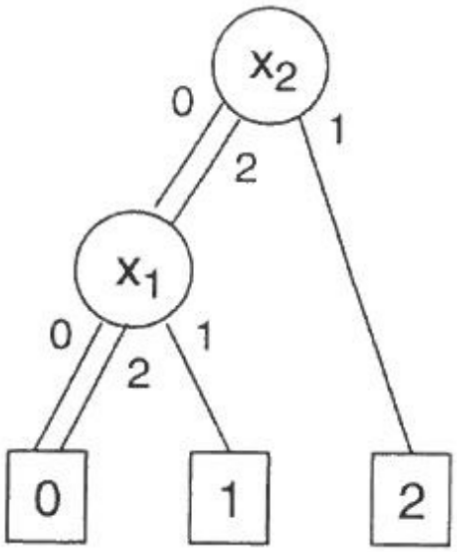
- Si spezza la funzione $f : M^n \rightarrow M$, rispettandone tutti i valori $i \in M$, generando m funzioni multi valore, input e output multi valore utilizzando la seguente logica $f^i : M^n \rightarrow \{0, 1\}$
- Infine si rappresenta ognuna delle funzioni spezzate nella notazione. [7]

4.1.2.2 Multiple Valued Network

Una Multiple Valued network è un grafo multi-livello simile a quello booleano con la differenza che ogni nodo rappresenta una funzione multi valore. Per sistemi con alto livello di astrazione l'utilizzo di questi grafi può rendere il design della progettazione più intuitivo. Prendiamo, per esempio, il controller di un semaforo, esso può prendere valore *rosso*, *giallo* o *verde* piuttosto che considerare l'encoding in modo binario. Questo tipo di progettazione è possibile solamente semplificando il sistema fino al punto in cui non è possibile renderlo ancora più ottimale nella manipolazione del dominio dei valori. Il lato negativo di questa tecnica è che la rappresentazione di un sistema non si presenta in forma canonica, poichè una funzione può assumere più rappresentazioni. Questo lato negativo rende il testing più difficoltoso.

4.1.2.3 MDDS

I Multiple Valued Decision Diagrams sono una generalizzazione degli alberi binari di decisione (BDDs). Le funzioni sono rappresentate creando un diagramma aciclico con nodi non terminali v etichettati da una variabile di indice $index(i), i \in \{1, 2, ..., n\}$. Ogni vertice v ha m uscite dirette verso i nodi figli, chiamati $child_j(v), j \in M$. Ogni vertice terminale v ha come attributo un valore $value(v)$ in M.



4.2 Studio del circuito binario

Analizzando i vari circuiti per testare la tesi è stato necessario, ai fini dell'analisi, sviluppare una tecnica di conversione dei circuiti binari di partenza. La caratteristica di una naturale rappresentazione di un circuito consiste nella sua divisione in nodi, ognuno dei quali viene rappresentato da una tabella di verità i cui input e output vanno a correlarsi tra loro tramite dei collegamenti. Soffermandosi più attentamente sulle tabelle di verità è stato possibile notare come, andando ad applicare un raggruppamento di n bit per ogni riga della tabella, sarebbe possibile riuscire a rappresentare un valore di dominio più grande di quello binario del circuito in analisi. Di conseguenza, così facendo, è possibile attuare una riduzione del numero di input richiesti dal gate.

$$\begin{array}{ccc} \underbrace{10} & \underbrace{11} & \underbrace{00} \\ \downarrow & \downarrow & \downarrow \\ 2 & 3 & 0 \end{array} \quad mod_3$$

Con questo approccio è possibile non snaturare quella che è la logica della tabella originale poiché, sapendo qual è il nuovo dominio applicato, vi è la possibilità di invertire il processo di conversione, ritornando quindi al circuito originale.

È necessario però considerare i vincoli presentati da questo approccio di conversione dei circuiti booleani. Queste limitazioni risiedono nel fatto che, al momento, la conversione può essere attuata solo e soltanto con un numero pari di input e che il numero di output sia un divisore del numero di bit utilizzati per il raggruppamento.

Una volta definito il metodo di conversione e preso atto dei vincoli di questo tipo di approccio, abbiamo proseguito con la scelta del nuovo dominio da applicare ai circuiti, questo con l'obiettivo di trovare il giusto compromesso per ottenere i seguenti risultati:

- Una riduzione degli input che facesse diminuire il costo del circuito di ingresso rispetto al circuito di partenza.
- Trovare un dominio multi valore che risulti efficace nella riduzione dei costi. avere un dominio superiore a quello booleano ma troppo elevato significherebbe che, per ogni input, ogni parte dovrebbe inserire tanti valori di input quanti quelli del dominio richiesto e, se si sceglie un dominio troppo grande, nonostante la riduzione degli input ci si troverebbe comunque con un costo totale del circuito elevato. Si è deciso quindi di sperimentare 2 approcci alla scelta di questo valore, uno statico e uno dinamico.
  - **Statico**: per ognuno dei circuiti andremo ad applicare lo stesso valore di dominio, il valore scelto è 3, in seguito procederemo quindi a raggruppare ogni 2 valori per riga della tabella di verità;
  - **Dinamico**:ogni circuito verrà analizzato andando a trovare il numero di input e output, successivamente procedendo con il calcolo del valore di M.C.D. e:
    * Se il valore trovato consente alla tabella di verità convertita di avere almeno 2 valori di input, in quel caso si utilizza quel valore per raggruppare i bit delle righe, altrimenti il valore viene dimezzato.
    * Se il valore risultante è un numero che creerebbe un dominio troppo grande, in quel caso viene utilizzato il dominio 3, come nel caso statico.

Una volta definite tutte le logiche di conversione è stato necessario affrontare una seconda problematica: identificare un formato di file che rispettasse determinati requisiti, tra questi citiamo:

- un formato in grado di rappresentare i circuiti
- un formato che rispetti le esigenze per poter applicare tutte le elaborazioni sopra descritte
- un formato che possa essere scelto come standard di rappresentazione in modo tale che i risultati ottenuti possano essere utilizzati dai software di sintesi che andremo ad utilizzare successivamente durante l'elaborazione della sperimentazione.

Si è presentata dunque la necessità di una rappresentazione che avesse la possibilità di rappresentare i circuiti con output multipli sulla stessa linea all'interno della tabella di verità.

Tra i vari formati a disposizione, dopo averne analizzate le caratteristiche, la scelta è ricaduta sul formato `PLA`, già ampiamente utilizzato per la rappresentazione di circuiti. Per comprendere meglio questa scelta è necessario però andare a sviscerare le caratteristiche e la struttura del formato selezionato.

4.2.1 Analisi della struttura PLA

La struttura di un file PLA può essere così descritta:

```
.i 4
.o 2
.ilb x1 x2 y1 y2
.ob f1 f2
0--0 00
0001 01
0-11 --
1-11 01
0101 10
10-- 01
11-- 00
.end
```

Questo `PLA` preso come esempio rappresenta un circuito composto da una sola tabella di verità, la sintassi va a descrivere gli elementi nel seguente modo:

- `.i`: numero di valori di input;
- `.o`: numero di valori di output;
- `.ilb`: nomi e ordine delle variabili di input;
- `.ob`: nomi e ordine delle variabili di output.

Il resto del file riportato rappresenta la tabella di verità con i possibili valori che si ottengono combinando i valori di input_file per ottenere gli output. Il circuito sopra descritto presenta 4 variabili di input e 2 valori di output. Come possiamo osservare all'interno della tabella di verità non tutti i valori sono booleani, è infatti presente anche il valore "-", esso prende il nome di don't care e, all'interno del file, può assumere due significati diversi:

- Quando - è presente negli output di una riga significa che l'output per quella determinata riga non genera un valore interessante ai fini dell'analisi;
- Quando - è presente negli input significa che, in quella posizione, la variabile a cui fa riferimento può assumere un qualsiasi valore non andando ad influire sul valore di output che si andrà ad ottenere.

La presenza di questo nuovo valore, non presente nel caso dei circuiti booleani classici, ci presenta la necessità di analizzare e capire il metodo con cui andare a gestire questi don't care all'interno del circuito.

4.2.2 La gestione dei don't care durante la conversione

I dont't care sono una caratteristica dei file PLA che deve essere gestita in modo funzionale ai cambi di dominio che andremo ad eseguire all'interno delle tabelle di verità dei nostri circuiti. Questa gestione è di primaria importanza in quanto, se affrontata nel modo sbagliato, modificherebbe la logica dell'intero circuito, rendendo di conseguenza la conversione errata. Per fare più chiarezza riguardo il trattamento che va applicato a questa caratteristica dei PLA, è presentato l'esempio di un circuito semplice che ha come valore di dominio di conversione il valore 3. Il circuito avrà la seguente struttura:

```
.i 4
.o 2
.ilb x1 x2 y1 y2
.ob f1 f2
0--0 00
0001 01
0-11 --
1-11 01
0101 10
10-- 01
11-- 00
.end
```

Una volta stabilito il nostro nuovo dominio di conversione sappiamo quanti bit dobbiamo raggruppare per rappresentare il nostro nuovo dominio. Nel nostro caso, essendo il domino 3, il numero di bit per rappresentare il valore più grande di questo dominio, abbiamo bisogno di un raggruppamento a 2 bit. Procediamo quindi a scandire riga per riga il circuito, raggruppando i bit a blocchi di due sia per tutti i valori di input che per quelli di output.

```
0- -0 00
00 01 01
0- 11 --
1- 11 01
01 01 10
10 -- 01
11 -- 00
```

Una volta creati questi cluster di bit ci rendiamo conto di come i dont't care, in alcuni casi, ci rendano impossibile la conversione al nuovo dominio. Per fare in modo che questo problema venga risolto, sarà necessario andare a sviluppare questi valori di don't care in 'normali' valori booleani.

Non è però sempre necessario andare a sviluppare i don't care, per questo motivo si differenziano due possibili casistiche di intervento applicabili a questi casi: - Quando il numero di '-' è uguale al numero di bit del raggruppamento multi valore. Quando si presenta questa situazione non c'è la necessità di sviluppare i valori, sarà infatti sufficiente inserire al posto della coppia di bit un nuovo - rappresentante il don't care.

10	−−	01
	↓	
2	−	1

- Quando nel gruppo di bit raggruppati il numero di - è minore del numero di bit richiesti. Quando si presenta questa circostanza è necessario procedere con l'espansione del dont't care a valori di verità, questo in modo tale da poter affrontare correttamente la conversione nel nuovo dominio.

1−	11	01
	↓	
10	11	01
11	11	01
	↓	
2	3	1
3	3	1

Analizzando e applicando queste due differenti casistiche ci è dunque possibile sviluppare tutto il circuito binario e ottenere una conversione esatta. Il circuito convertito sarà quindi.

```
00 0
02 0
10 0
12 0
01 1
03 -
13 -
23 1
11 2
2- 1
3- 0
```

Con questa nuova struttura è in fine possibile procedere all'analisi dei circuiti, proseguendo in seguito con l'attività di sintesi logica.

4.3 Implementazione della conversione

La conversione dei circuiti in analisi viene implementata utilizzando il linguaggio di programmazione Python nella versione 3.8. Questo implemento avviene senza l'ausilio di nessun package esterno, vengono infatti utilizzate solamente librerie che sono già comprese nel linguaggio in questione.

Questa dinamica fa si che il sistema, una volta installato il linguaggio di programmazione, se non già presente all'interno del SO, risulti essere "ready to use".

4.3.1 Analisi del circuito

```
def read_pla(path_file):
  inp = None
  out = None
  inp_array = []
  out_array = []
  truth_table = []
  with open(path_file, 'r') as input_file:
    for line in input_file.readlines():
      if '.i' in line and line[2] == ' ':
        inp = line.split(' ')[1]
      elif '.o' in line and line[2] == ' ':
        out = line.split(' ')[1]
      elif '.ilb' in line:
        inp_array = line.strip().split(' ')[1:]
      elif '.ob' in line:
        out_array = line.strip().split(' ')[1:]
      elif '.end' in line:
        continue
      else:
        line = {
          'inp':  line.strip().split(' ')[0],
          'out':  line.strip().split(' ')[1]
        }
        truth_table.append(line)
  return inp, out, inp_array, out_array, truth_table
```

All'interno del processo sopra rappresentato la funzione prende in input un circuito binario in formato PLA. Riga per riga, procede ad un'analisi delle informazioni in base al prefisso all'interno di ogni riga del file. Questo prefisso va a differenziare le informazioni del circuito e scandendo il file recupera le seguenti informazioni:

- Tabella delle verità
- Numero di input
- Numero di output
- Array contenente il nome delle variabili di input;
- Array contenente il nome delle variabili di output;

Queste informazioni saranno necessarie successivamente per poter convertire il circuito e poterne calcolare costi effettivi.

4.3.2 Espansione dei don't care

Una volta ottenute tutte le informazioni disponibili dal circuito dato, succesivamente è necessario andare ad identificare all'interno delle tabelle di verità quali sono i don't care a cui bisogna espandere i valori e quelli che si possono ignorare. Questo passaggio nello specifico è stato la parte più impegnativa riguardante le funzionalità del programma.

```
n_dont_care = ''
  for i in range(dv):
    n_dont_care += '-'
```

Con questo ciclo vado a creare un array di - consecutivi che rispecchiano i cluster di don't care da ignorare nel durante l'analisi delle tabelle di verità. Avrò quindi un array di n - tanti quanti il numero di bit per rappresentare i numeri del nuovo dominio.

```
len_truth_table = len(truth_table)
i = 0
while i < len_truth_table:
  if '-' in truth_table[i]['inp']:
    truth_table[i]['inp'] = ''.join(truth_table[i['inp'])
    truth_table[i]['inp'] = [truth_table[i]['inp'[a:a+dv]
                             for a in range(0, le(truth_table[i]['inp'], dv)]
    for a in range(len(truth_table[i]['inp'])):
        if truth_table[i]['inp'][a] == n_dont_care:
            truth_table[i]['inp'][a] = 'k'*le(n_dont_care)
    truth_table[i]['inp'] = ''.join(truth_table[i['inp'])
    new_lines = resolve_dont_care(truth_table[i],'inp')
    truth_table = truth_table[:i] + new_lines +truth_table[i+1:]
    len_truth_table = len(truth_table)
  i += 1
```

Tramite questo ciclo *while* l'intero circuito viene scandito e:

- I valori di input vengono divisi in cluster di n bit per rappresentare i nuovi valori del dominio;

```
truth_table[i]['inp'] = [
   truth_table[i]['inp'][a:a+dv]
   for a in range(0, le(truth_table[i]['inp'], dv)]
 ]
```

- Se all'interno dei cluster sono presenti solamente don't care il valore - viene sostituito con il valore k per differenziarli dai valori dont't care che successivamente verranno espansi

```
if truth_table[i]['inp'][a] == n_dont_care:
  truth_table[i]['inp'][a] = 'k'*le(n_dont_care)
```

Ora la tabelle è finalmente pronta per l'espansione dei don't care tramite l'appostia funzione sviluppata

Si è utilizzato un ciclo *while* in questa funzione nonostante in Python sia meno efficiente a causa della natura della sua implementazione; questo perchè la lunghezza della tabella della verità aumenta dinamicamente con l'andare dell'espansione dei don't care individuati.

4.3.2.1 Resolve don't care

La funzione inizia andando ad individuare quanti don't care sono presenti nella linea

```
def resolve_dont_care(line, in_out):
  n_dc = pow(2, line[in_out].count('-'))
```

Successivamente viene iniziato un ciclo che scandisce ogni elemento della linea della tabella di verità, viene creato un array binario della di tante righe quante saranno dopo l'espansione dei don't care.

```
for h in range(line[in_out].count('-')):
  val_array = create_0_1_array(n_dc, pow(2, h))
```

Successivamente vengono differenziate 2 casistiche:

- Prima iterazione

```
if h == 0:
  for i in range(n_dc):
    it = 0
    new_line = []
    for j in range(len(input_entry)-1, -1, -1):
      if input_entry[j] == '-' and it == 0:
        new_line.append(val_array[(len(val_array)-1) - i])
        it += 1
      else:
        new_line.append(line[in_out][j])
    new_array.append(new_line[::-1])
```

- Tutte le successive

```
else:
  it = 0
  for c, l in enumerate(new_array):
    for j in range(len(l)-1, -1, -1):
      if l[j] == '-' and it == 0:
        l[j] = val_array[(len(val_array)-1) - c]
        it += 1
    it = 0
```

Ogni riga viene scandita a partire dal fondo per iniziare a convertire i don't care corrispondenti ai valori meno significativi. Sfruttando l'array binario creato sopra andiamo a sostituire il simbolo '-' con con il valore booleano corrispondente e quindi a creare una nuova linea di un array che conterrà tutti i valori ottenuti dall'espansione.

```
if in_out == 'inp':
  return [{'inp':  l, 'out':  line['out']} for l in new_array]
else:
  return [{'inp':  line['inp'], 'out':  l} for l in new_array]
```

Come possiamo notare dal `return` della funzione, questa operazione di espansione è possibile sia sugli input che su gli output. Come detto nel capitolo precedente anche l'espansione dei valori di don't care degli output è importante per non perdere l'integrità del circuito.

L'oggetto restituito dalla funzione è un array contenente un vocabolario con la stessa struttura che è stata usata durante l'estrazione dei dati dal file `PLA`, questo array sostituirà la riga contenente i don't care espansi.

4.3.2.2 create_0_1_array

```
def create_0_1_array(le, pad):
  count_1 = pad
  count_0 = pad
  return [str(1) if i % (count_1 + count_0) < count_1
    else str(0) for i in range(le)]
```

Questa funzione restituisce una tabella di verità data una linea con all'interno un numero di don't care che necessitano di espansione.

Questa tabella creata verrà inserita al posto della linea con i don't care del circuito in modo tale da avere la tabella corretta per la conversione.

Una volta terminato l'intero ciclo tutti i dont't care saranno risolti e la tabella sarà pronta per la conversione nel nuovo valore di dominio.

4.4 La conversione in multi valore

Le funzioni analizzate e osservate nelle sezioni precedenti restituiscono sempre come valore di output una matrice corrispondente alla tabella della verità sviluppata. Questa tabella verrà utilizzata dalla funzione di conversione seguendo lo schema spiegato all'inizio del capitolo.

```
def create_mv_truth_table(truth_array, dv):
  conv_truth = []
  for line in truth_array:
    line['inp'] = ''.join(line['inp'])
    line['inp'] = [line['inp'][i * dv:(i + 1) * dv]
      for i in range((len(line['inp']) + dv - 1) // dv)]
    if '-' not in line['out']:
      line['out'] = ''.join(line['out'])
      line['out'] = [line['out'][i * dv:(i + 1) * dv]
        for i in range((len(line['out']) + dv - 1) // dv)]
      l_supp_inp = []
      l_supp_out = []
      for val in line['inp']:
        if '-' not in val:
          l_supp_inp.append(int(val, 2))
        else:
          l_supp_inp.append('-')
      for val in line['out']:
        l_supp_out.append(int(val, 2))
      conv_truth.append({
        'inp':  l_supp_inp,
        'out':  l_supp_out
      })
  return conv_truth
```

La funzione prende in ingresso la matrice espansa creata precedentemente e il numero di bit da utilizzare per rappresentare il massimo numero del nuovo dominio multi valore.

Inizialmente la funzione 'spezza' la stringa dei valori di input in gruppi di tanti elementi quanti sono i bit per rappresentare il massimo numero, successivamente viene controllato se il gruppo è composto da:

- dont't care: in questo caso si procede sostituendoli con un singolo simbolo '-'
- numeri binari: in quest'altro caso si procede con la conversione tramite il metodo int(), per applicare questa conversione sono però necessari due parametri differenti:
  - un numero o una stringa di numeri da convertire
  - la base a cui si vuole fare la conversione

Una volta scandita tutta la tabella di verità abbiamo finalmente ricavato un circuito convertito. L'utlimo processo da mettere in atto consiste nell'andare a creare un file compatibile per la sintesi dei circuiti.

4.5 La creazione del file blfmv

Per i circuiti multi valore non viene adottato il formato PLA in quanto quest'ultimo viene utilizzato solamente per i circuiti binari. Per sopperire a questa limitazione data dal formato PLA si farà uso di un altro formato standard appositamente creato per la logica multi valore: il formato blfmv.

Questo formato deriva dal formato blif, ampiamente utilizzato per la logica binaria.

```
import string
mv_input = [i for i in list(string.ascii_lowercase)[
    :len(mv_table[0]['inp'])]
  ]
mv_output = ['o{}'.format(i) for i in range(len(mv_table[0]['out']))]
with open('{}/blfmv/{}.mv'.format(working_dir, nomefile), 'w') as blif:
  blif.write('.model {}\n'.format(working_dir, nomefile))
  blif.write('.inputs {}\n'.format(
      ' '.join(map(str, mv_input))))
  blif.write('.outputs {}\n'.format(
      ' '.join(map(str, mv_output))))
  blif.write('.mv {} {}\n'.format(
      ', '.join(map(str, mv_input)), mv))
  blif.write('.mv {} {}\n'.format(
      ', '.join(map(str, mv_output)), mv))
  for count, out in enumerate(mv_output):
      blif.write('.table {} {}\n'.format(' '.join(map(str,mv_input)), out))
    for line in mv_table:
```

```
      blif.write('{} {}\n'.format(
        ' '.join(map(str, line['inp'])), line['out'][count]))
  blif.write('.end\n')
```

Questa funzione crea un file .mv relativo al circuito creato. Scegliamo questa tipo di file e sintassi poiché nelle fasi successive andremo ad utilizzare dei software che accettano questa sintassi per valutare e sintetizzare i circuiti.

4.6 La sintesi

Una volta ottenuti sia il circuito booleano che quello multi valore procederemo con la sintesi dei circuiti. Quest'ultima si pone come obiettivo la riduzione e l'ottimizzazione della struttura dei circuiti stessi, andando a diminuire il numero di ingressi e di tabelle della verità in modo tale da avere dei costi totali minori.

I programmi utilizzati per la sintesi logica sono stati entrambi sviluppati dall'università di Berkley e attualmente sono disponibili con licenza open source.

Per la sintesi durante l'analisi sperimentale sono state utilizzate due alternative:

- MVSIS
- ABC

4.6.1 MVSIS

E' il primo programma utilizzato, al suo interno troviamo diversi metodi di sintesi da poter applicare, i quali sfruttano tecniche differenti. La particolarità di questo tool consiste nel fatto che il programma è in grado di accettare come input sia circuiti binari che multi valore. Il lato negativo dell'utilizzo di questo programma, d'altra parte, è che l'ultima versione di questo software risale al 2005 e quindi non è più mantenuto.

4.6.2 ABC

ABC, come la sua alternativa descritta sopra, mette a disposizione delle tecniche di sintesi che presentano il vantaggio di avere degli script pronti che uniscono più metodi, in questo modo è possibile avere la certezza di non commettere errori o di utilizzare una concatenazione di metodi inefficace.

ABC è un'evoluzione di MVSIS e del più vecchio SIS. Esso viene tuttora mantenuto ma, a differenza di MVSIS, se gli viene dato in ingresso un valore di input, esso viene successivamente convertito in binario e solo in seguito è possibile utilizzare i metodi di sintesi.

Una caratteristica molto utile di questi programmi sta nel fatto che essi accettano come parametri di ingresso degli script contenenti tutte le istruzioni da eseguire. Proseguendo con lo sviluppo della tesi sarà possibile dimostrare come tutto il processo di analisi viene automatizzato sfruttando questa caratteristica.

4.6.3 La sintesi tramite i software

La letteratura mette già a disposizione delle sequenze di metodi di sintesi e pulizia dei circuiti efficace ed efficiente, questi comandi si possono chiamare semplicemente dando in input ai due programmi un file contenente un alias per questa sequenza di istruzioni.

```
source abc_alias.abc
read_blif_mv ./prova/blfmv/amd.mv
strash
compress2
cl
resyn2
cl
write_blif_mv ./prova/blfmv/synth/amd.mv
```

Il programma, in questo caso ABC, non fa altro che prendere come input uno dei circuiti creati precedentemente e, applicando i metodi di sintesi, crea un ulteriore file contenente il circuito sintetizzato. In questo modo è successivamente possibile fare degli ulteriori confronti, una volta sintetizzati tutti i circuiti.

4.7 Il calcolo dei costi del circuito

Il costo del circuito viene calcolato sulla base di due fattori fondamentali: - Quanti input devono inviare entrambe le parti - Quanti valori devono inviare per ciascun valore di input in base al dominio

Per calcolare questo tipo di informazioni si devono effettuare delle ulteriori operazioni di lettura sui file precedentemente creati, il tutto analizzando due differenti caratteristiche:

- Gli ingressi di ogni tabella di verità del circuito (andando a controllare quali dei valori di input devono essere inseriti da una delle parti e quali sono ingressi di valori di output delle altre tabelle.)
- Il dominio dei valori di input che devono inserire le parti.

4.7.1 blfmv vs pla

Per poter eseguire il confronto sopra citato si è preferito utilizzare file che avessero la stessa sintassi di rappresentazione del circuito, sia nel caso multi valore che in quello booleano.

Il formato *pla* attualmente non dispone di un'alternativa per i file multi valore, ma non è ancora supportata dai tool che abbiamo in utilizzo. Per questo motivo si è deciso di utilizzare il formato *blif* per rappresentare i circuiti binari. *Blif* è l'alternativa binaria a *blfmv*, la sintassi è la stessa con la differenza che non viene specificato il dominio dei valori di input in quanto sempre booleano.

La differenza con *pla*, invece, sta nel fatto che gli output non possono essere in numero maggiore di 1 per tabella: nella rappresentazione avrò quindi, per ogni nodo del circuito, tante tabelle quanti sono gli output di quel nodo. Questa tipologia di rappresentazione è utilizzata anche nei file *blfmv*, quindi sarà possibile una comparazione 1:1 tra il circuito binario e quello multi valore.

Per effettuare questa conversione vengono in aiuto i tool ABC e MVSIS, poiché entrambi contengono metodi di conversione automatica da *pla* a *blif*. Questa operazione viene effettuata tramite uno script contenente le istruzioni da eseguire e dato come parametro di ingresso al programma. [11]

```
read_pla ./prova/pla/alu2.pla
write_blif ./prova/blif/alu2.blif
```

A questo punto dell'analisi abbiamo a disposizione tutti i file per poter fare effettuare il confronto dei costi.

4.7.2 Implementazione

```
with open('{}/{}'.format(working_dir, circuito)) as circ:
  input = None
  output = None
  mv = int(0)
  table_array = []
  for line in circ.readlines():
    if '.inputs' in line.strip():
      input = line.strip().split(' ')[1:]
    if '.outputs' in line.strip():
      output = line.strip().split(' ')[1:]
    if '.mv' in line.strip():
      if mv < int(line.strip().split(' ')[-1]):
        mv = int(line.strip().split(' ')[-1])
    if '.table' in line.strip():
      l = line.strip().split(' ')
      table_array.append(
        {
          'input':    l[1:len(l)-1],
          'output':   l[-1]
        }
      )
    if '.names' in line.strip():
      l = line.strip().split(' ')
      table_array.append(
        {
          'input':    l[1:len(l)-1],
          'output':   l[-1]
        }
      )
  # i blif non hanno .mv, quindi gli do il valore di dominio
  if mv == 0:
    mv = 2
  return {
    'dominio':   mv,
    'input':    input,
    'output':   output,
    'tabelle':  table_array
  }
```

Questa funzione, dato un circuito sia binario che mutli-valore, prende tutte le informazioni ad essa necessarie per poter effettuare il calcolo.

```
costo = 0
for t in circuito['tabelle']:
  intersection = len(set(circuito['input']).intersection(t['input']))
  costo = costo + pow(circuito['dominio'], intersection)
return costo
```

Una volta eseguita questa conversione è stato possibile ottenere tutti i valori di costo sia dei circuiti booleani che di quelli binari.

```
with open('calcolo_costi.csv', 'a') as file:
  file.write(
    'NOME CIRC;COSTO BOOLEANO;INPUT ALICE;INPUT BOB;
    DOMINIO MULTIVALORE;COSTO MULTIVALORE;INPUT ALICEINPUT BOB\n')
  for blfmv in listdir('{}/blfmv'.format(working_dir)):
    if blfmv.endswith('.mv'):
      circ_mv = info_circuito(
          '{}/blfmv/synth'.format(working_dir),blfmv)
      costo_mv = calcolo_costo_circuito(circ_mv)
      if len(circ_mv['input']) % 2 != 0:
          alice_var_mv = int(len(circ_mv['input'])/ 2)
          bob_var_mv = int(len(circ_mv['input']) /2) + 1
      else:
          alice_var_mv = int(len(circ_mv['input'])/ 2)
          bob_var_mv = int(len(circ_mv['input']) /2)
      circ_bool = info_circuito(
          '{}/blif/synth'.format(working_dir), '{}blif'.format(blfmv.split('.')[0]))
      costo_bool = calcolo_costo_circuito(circ_bool)
      if len(circ_bool['input']) % 2 != 0:
          alice_var_bool = int(len(circ_boo['input']) / 2)
          bob_var_bool = int(len(circ_bool['input') / 2) + 1
      else:
          alice_var_bool = int(len(circ_boo['input']) / 2)
          bob_var_bool = int(len(circ_bool['input') / 2)
      file.write('{};{};{};{};{};{};{};{}\n'.forma(blfmv.split('.')[
          0], costo_bool, alice_var_bool,bob_var_bool, circ_mv['dominio'],costo_mv, alice_var_mv,
```

Ottenute tutte le informazioni, per avere una migliore visione per l'analisi, esse vengono salvate all'interno di una file *CVS* con la funzione sopra mostrata.

4.8 Automatizzazione dei processi

Tutte le fasi mostrate sopra, più altre secondarie, sono state automatizzate in modo tale che:

- non si debbano effettuare operazioni ridondati per ogni circuito
- non si generino perdite di dati causate da errori umani.

4.8.1 Struttura

```
.
  abc_alias.abc           #alias con metodi sintesi ABC
  automate.py             #automatizzatore dei processi
  calcolo_costi.csv
  calcolo_costi_no_synth.csv
  facili  #directory contenente i file d'analisi
     blfmv
        synth # circuiti sintetizzati
     blif
        synth
     pla_extended #pla espansi per debugging
  lib_mv  # libreria con tutti le funzioni create
     automate_mvsis.py
     bin_to_mv.py
     caloclo_costo.py
  pla_to_blif.mvsis   #file con sintesi per MVSIS
  synth_bool.abc      #file di sintesi  binaria per ABC
  synth_mv.abc        #file di sintesi multi valore per ABC
```

```
working_dir = './prova'
 if path.exists('{}/blif'.format(working_dir))
    and path.exists('{}/blfmv'.format(working_dir)):
        rmtree('{}/blif'.format(working_dir))
        rmtree('{}/blfmv'.format(working_dir))

makedirs('{}/blif/synth'.format(working_dir))
makedirs('{}/blfmv/synth'.format(working_dir))
makedirs('{}/blfmv/synth/abc'.format(working_dir))
makedirs('{}/blfmv/synth/mvsis'.format(working_dir))

if path.exists('{}/synth_out.mvsis'.format(working_dir)):
    remove('{}/synth_out.mvsis'.format(working_dir))

if not path.exists('{}/pla'.format(working_dir)):
    print('CARTELLA PLA NON PRESENTE')
    exit
```

In questa fase viene creata la struttura delle directory che conterranno i file con i circuiti sintetizzati. Nel caso in cui questa struttura fosse già presente, verranno cancellate tutte le cartelle e i file all'interno e successivamente ricreate le cartelle vuote.

```
bin_to_mv_mcd(working_dir)
if exists('./pla_to_blif.mvsis'):
  remove('./pla_to_blif.mvsis')
pla_to_blif(working_dir)
```

In questa sezione vengono chiamate due utility create con lo scopo di generare dei file utilizzabili sia da **ABC** che da **MVSIS**.

- **bin_to_mv_mcd**: ha lo scopo di creare i file *blfmv* di ogni circuito all'interno della directory di lavoro
- **pla_to_blif**: prende dalla directory di lavoro i file *pla* contenenti i circuiti e li converte in *blif* per avere un formato identico di rappresentazione tra i circuiti binari e quelli multi valore.

4.8.1.1 pla_to_blif

```
def pla_to_blif(working_dir):
  with open('./pla_to_blif.mvsis', 'w') as file:
    file.write("source abc_alias.abc\n")
    for ele in os.listdir('{}/pla'.format(working_dir)):
      file.write('read_pla {}/pla/{}\n'.format(working_dir,ele))
      file.write('write_blif {}/blif/{}.blif\n\n'.format(
        working_dir, ele.split('.')[0]))
```

Questa funzione restituisce un file leggibile come input da *mvsis* o *abc* in grado di cambiare automaticamente formato dei file da *pla* a *blif*. In questo modo la conversione viene fatta automaticamnte da uno dei programmi scelti senza ricorrere ad ulteriore codice o incappare in errori di conversione. Il file ottenuto ha la seguente struttura.

```
source abc_alias.abc
read_pla ./prova/pla/alu2.pla
write_blif ./prova/blif/alu2.blif

read_pla ./prova/pla/br2.pla
write_blif ./prova/blif/br2.blif

read_pla ./prova/pla/amd.pla
write_blif ./prova/blif/amd.blif
```

In questo modo ogni circuito viene convertito semplicemente chiamando

```
call(['abc', '-F', './pla_to_blif.mvsis'])
```

4.8.1.2 bin_to_mv_mcd

```
def bin_to_mv_mcd(working_dir):
  for pla in listdir('{}/pla'.format(working_dir)):
    if pla.endswith('.pla'):
      inp, out, inp_array, out_array, truth_table = read_pla(
        '{}/pla/{}'.format(working_dir, pla))
      dv = euclide(int(inp), int(out))
      if int(inp.strip()) == int(out.strip()) or dv == int(inp):
        dv = dv/2
      if dv >= 8:
        dv = 2
        mv = pow(2, dv)
        print('{}\ninp: {}\nout:{}\nmulti-valore:{}\ndivide ogni:{}\n'.format(pla,
          inp.strip(), out.strip(), int(mv), int(dv)))
        truth_table = expand_dont_care(truth_table, int(dv))
        mv_table = create_mv_truth_table(truth_table, int(dv))
        crete_blif_mv(working_dir, mv_table, int(mv),
          basename(pla).split('.')[0])
```

In questa funzione vengono racchiuse le funzioni descritte precedentemente per la conversione del circuito da binario a multi valore e la creazione dei file utilizzabili dai motori di sintesi. Come descritto nei capitoli precedenti verranno utilizzate due tecniche di conversione in base al circuito, una statica e una dinamica. Di seguito verrà mostrata solo la soluzione dinamica in quanto più complessa ma dalla struttura analoga alla proposta statica.

```
dv = euclide(int(inp), int(out))
  if int(inp.strip()) == int(out.strip())
    or dv == int(inp):
    dv = dv/2
  if dv >= 8:
    dv = 2

def euclide(a, b):
  while(b != 0):
    R = a % b
    a = b
    b = R
  return a
```

In questa porzione avvengono le fasi di individuazione del M.C.D. tramite la funzione *euclide* e viene effettuato il controllo nel caso in cui il dominio trovato fosse troppo grande o che ci fosse un solo input utilizzando questo dominio.

```
synth(working_dir)

call(['abc', '-F', './synth_bool.abc'])
call(['abc', '-F', './synth_mv.abc'])
call(['mvsis', '-F', './synth_mv.mvsis'])
```

L'ultimo passo consiste nella sintesi dei circuiti ottenuti e nella loro controparte binaria, la funzione `synth` fa in modo che vengano creati 2 file da dare in ingresso ad `MVSIS` o `ABC` che sintetizzino tutti i circuiti e salvino il risultato in dei file.

4.8.1.3 Synth

```
def synth(working_dir):
  with open('./synth_bool.abc', 'w') as file:
    file.write("source abc_alias.abc\n")
    for ele in os.listdir('{}/blif'.format(working_dir)):
      if ele.endswith('.blif'):
        file.write('read_blif {}/blif/{}\n'.format(working_dir,ele))
        file.write('cl\n')
        file.write('resyn2\n')
        file.write('cl\n')
        file.write('write_blif {}/blif/synth/{}.blif\n\n'.format(working_dir,ele.split('.')[0]))
  with open('./synth_mv.abc', 'w') as file:
    file.write("source abc_alias.abc\n")
    for ele in os.listdir('{}/blfmv'.format(working_dir)):
```

```
      if ele.endswith('.mv'):
        file.write('read_blif_mv {}/blfmv/{}\n'.format(working_dir, ele))
        file.write('strash\n')
        file.write('compress2\n')
        file.write('cl\n')
        file.write('resyn2\n')
        file.write('cl\n')
        file.write('write_blif_mv {}/blfmv/synth/abc/{}\n\n'.format(working_dele))

  with open('./synth_mv.mvsis', 'w') as file:
    for ele in os.listdir('{}/blfmv'.format(working_dir)):
      if ele.endswith('.mv'):
        file.write('read_blif_mv {}/blfmv/{}\n'.format(working_dir, ele))
        file.write('sweep\n')
        file.write('eliminate -l 1\n')
        file.write('simplify -m nocomp\n')
        file.write('eliminate -l 1\n')
        file.write('sweep\n')
        file.write('eliminate -l 5\n')
        file.write('simplify\n')
        file.write('sweep\n')
        file.write('eliminate -l 1\n')
        file.write('sweep\n')
        file.write('fullsimp -m nocomp\n')
        file.write('write_blif_mv {}/blfmv/synth/mvsis/{}\n\n'.format(working_dir, ele
```

- **synth_bool**: crea il file per la sintesi utilizzata da **ABC** per i circuiti booleani;
- **synth_mv.abc**: crea il file per la sintesi utilizzata da **ABC** per i circuiti multi valore;
- **synth_mv**: crea il file per la sintesi utilizzata da **MVSIS** per i circuiti multi valore;

Viene poi controllato se è già presente un file contenente dei costi e, se c'è, lo elimina

```
if exists('./calcolo_costi.csv'):
    remove('./calcolo_costi.csv')
if exists('./calcolo_costi_no_synth.csv'):
    remove('./calcolo_costi_no_synth.csv')
```

Infine i costi dei circuiti possono essere calcolati

```
calcolo_costi_synth(working_dir)
calcolo_costi_no_synth(working_dir)

def calcolo_costo_circuito(circuito):
  costo = 0
  for t in circuito['tabelle']:
    intersection = len(set(circuito['input']).intersection(t['input']))
    costo = costo + pow(circuito['dominio'], intersection)
  return costo
```

Le operazioni di calcolo del costo vengono effettuate e, infine, viene generato il file *CSV* contenente i risultati.

Chapter 5

Risultati sperimentali

Una volta concluso il processo di conversione dei circuiti e terminato tutto il sistema di automazione dei processi, si è proceduto con l'analisi sperimentale dei dati raccolti, in modo tale da valutare se questo nuovo approccio sviluppato potesse essere considerato una alternativa possibile o vincente rispetto alle proposte già presenti in letteratura.

Tutte le operazioni computazionali sono state effettuate utilizzando un DELL-XPS 13 9350, la macchina ha le seguenti specifiche:

- Processore: Intel i5-6200U@2.30GHz con 2 core e 4 thread
- Memoria: 8Gb LPDDR3@1866MHz
- Sistema Operativo: Ubuntu 21.04

5.1 MVSIS

Nelle prime fasi di sperimentazione si e' decido di utilizzare MVSIS come motore di sintesi, sia per i circuiti binari che per quelli multi valore. Questo primo approccio è nato dalla volontà di replicare le operazioni che si svolgono solitamente sulla logica binaria per poi applicarle alla logica multi valore.

Per le operazioni di sintesi si sono utilizzati i seguenti comandi:

```
read_blif_mv ./prova/blfmv/amd.mv
sweep
eliminate -l 1
simplify -m nocomp
eliminate -l 1
sweep
eliminate -l 5
simplify
sweep
eliminate -l 1
sweep
fullsimp -m nocomp
write_blif_mv ./prova/blfmv/synth/mvsis/amd.mv
```

Con questi comandi si va ad attuare una sintesi del circuito preso in ingresso, nello specifico vengono effettuate tutte le operazioni necessarie di pulizia e di rimozione di nodi inutili tramite i comandi *eliminate* e *sweep*. Successivamente, tramite *simplify* e *fullsimp*, il circuito viene semplificato su due livelli: sia per quanto concerne i nodi presenti all'interno del circuito, sia per quanto riguarda il circuito stesso nella sua interezza.

Questi comandi sono la conversione della variante **MVSIS** di *script.rugged* usato con il tool per la logica binaria SIS. In questo caso, però, Non è stato possibile operare una conversione 1:1 dello script in quanto non tutti i comandi sono replicabili.

Per far fronte alla mancanza di replicabilità di alcuni comandi in questo sistema, è stato necessario eseguire un'ulteriore ricerca Sia in rete che in altri paper riguardanti l'argomento. Quello che ne viene dedotto è che è stato creato ed esiste un determinato script, chiamato *mvsis.rugged* che, secondo la nostra ricerca, dovrebbe rappresentare la vera conversione adattata per il multi valore. Purtroppo però non è stato possibile in alcun modo reperire questo file in nessun repository, rendendolo quindi per noi non utilizzabile ai fini delle analisi.

5.1.1 Conversione multi valore fissa

Come primo approccio alla conversione da binaria a multi valore abbiamo optato per lo stesso valore di dominio per tutti i circuiti. La scelta è stata quella di andare ad aggiungere altri 2 valori rispetto a ad un circuito binario utilizzando il modulo 3.

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	DOMINIO MULTI	COSTO MULTI	INPUT ALICE	INPUT BOB	% Guadagno
amd	469	7	7	3	47568	3	4	-10042%
tms	322	4	4	3	2048	2	2	-536%
pdc	2818	7	7	3	1310720	4	4	-46412%
mlp4	594	4	4	3	784	2	2	-32%
apla	343	5	5	3	1029	2	3	-200%
f51m	261	4	4	3	340	2	2	-30%
m4	974	4	4	3	1601	2	2	-64%
newtpla2	48	5	5	3	2048	2	3	-4167%
test1	1095	4	4	3	1280	2	2	-17%
m2	339	4	4	3	1121	2	2	-231%
br2	179	6	6	3	16384	3	3	-9053%
alu1	83	6	6	3	5122	3	3	-6071%
sqr6	197	3	3	3	276	1	2	-40%
bench	198	3	3	3	256	1	2	-29%
in5	446	7	7	3	117440512	6	6	-26331853%
newtpla1	33	5	5	3	1024	2	3	-3003%
m3	571	4	4	3	1169	2	2	-105%
newapla	83	6	6	3	9728	3	3	-11620%
l8err	454	4	4	3	337	2	2	26%
t4	238	6	6	3	10240	3	3	-4203%
br1	252	6	6	3	16384	3	3	-6402%
fout	442	3	3	3	320	1	2	28%

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	DOMINIO MULTI	COSTO MULTI	INPUT ALICE	INPUT BOB	% Guadagno
mp2d	136	7	7	3	7	3	4	95%
alu2	225	5	5	3	1282	2	3	-470%
t3	131	6	6	3	4	3	3	97%
p3	385	4	4	3	1408	2	2	-266%
m1	125	3	3	3	225	1	2	-80%
bcd_div3	54	2	2	3	32	1	1	41%
alu3	139	5	5	3	4	2	3	97%
in7	182	7	7	3	23085056	6	7	-12683997%
	393,87	**4,87**	**4,87**		**4731943,63**	**2,43**	**2,83**	**-13039,51**
	COSTO BOOL MEDIO				**COSTO MULTI MEDIO**			

Questa tabella racchiude i risultati ottenuti dalla sintesi dei circuiti multi valore e dei circuiti binari. Possiamo notare che nella maggior parte dei casi il circuito binario mantiene ancora una grande efficienza rispetto a quello multi valore.

5.1.2 MVSIS e dominio variabile

Nella sperimentazione successiva si è voluto testare un dominio variabile in base al MCD, come descritto nel capitolo precedente. Nella seguente tabella vengono mostrati i costi dei circuiti senza sintesi. Possiamo osservare che nella maggior parte dei casi il dominio multi valore senza sintesi ha un costo minore. Questo risultato può essere considerato una conseguenza diretta del minor numero di input che le parti dovranno inserire all'interno del protocollo durante la computazione. Questi input, essendo in minor quantità, rendono l'intero processo di sintesi più leggero e meno costoso.

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	DOMINIO MULTI	COSTO MULTIVALORE	INPUT ALICE	INPUT BOB	% GUADAGNO
amd	393216	7	7	4	196608	3	4	50%
tms	4096	4	4	16	1024	1	1	75%
pdc	2621440	8	8	4	1310720	4	4	50%
mlp4	2048	4	4	16	512	1	1	75%
apla	12288	5	5	4	6144	2	3	50%
f51m	2048	4	4	16	512	1	1	75%
m4	4096	4	4	16	1024	1	1	75%

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	DOMINIO MULTI	COSTO MULTIVALORE	INPUT ALICE	INPUT BOB	% GUADAGNO
newtpla2	4096	5	5	4	2048	2	3	50%
test1	2560	4	4	4	1280	2	2	50%
m2	4096	4	4	16	1024	1	1	75%
br2	32768	6	6	16	8192	1	2	75%
alu1	32768	6	6	16	8192	1	2	75%
sqr6	705	3	3	8	256	1	1	64%
bench	512	3	3	4	256	1	2	50%
in5	1835008	9	9	4	117440512	6	6	-6300%
newtpla1	2048	5	5	4	1024	2	3	50%
m3	4096	4	4	16	1024	1	1	75%
newapla	40960	6	6	4	20480	3	3	50%
l8err	1283	4	4	16	512	1	1	60%
t4	32768	6	6	16	8192	1	2	75%
br1	32768	6	6	16	8192	1	2	75%
fout	640	3	3	4	320	1	2	50%
mp2d	229376	7	7	128	32768	1	1	86%
alu2	8192	5	5	4	4096	2	3	50%
t3	32768	6	6	16	8192	1	2	75%
p3	3074	4	4	4	1792	2	2	42%
m1	768	3	3	8	256	1	1	67%
bcd_div3	64	2	2	4	32	1	1	50%
alu3	8192	5	5	4	4096	2	3	50%
in7	1310720	9	9	4	335544320	6	7	-25500%
	221982,07	5,03	5,03		15153786,67	1,80	2,27	-10,02
	COSTO BOOL MEDIO				COSTO MULTI MEDIO			

Una vota applicato la sintesi dei circuiti i risultati sono i seguenti:

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	DOMINIO MULTI	COSTO MULTIVALORE	INPUT ALICE	INPUT BOB	% GUADAGNO
amd	469	7	7	4	47568	3	4	-10042%
tms	322	4	4	16	1024	1	1	-218%
pdc	2818	7	7	4	1310720	4	4	-46412%
mlp4	594	4	4	16	512	1	1	14%
apla	343	5	5	4	1029	2	3	-200%
f51m	261	4	4	16	272	1	1	-4%
m4	974	4	4	16	1024	1	1	-5%
newtpla2	48	5	5	4	2048	2	3	-4167%
test1	1095	4	4	4	1280	2	2	-17%
m2	339	4	4	16	1024	1	1	-202%
br2	179	6	6	16	8192	1	2	-4477%
alu1	83	6	6	16	4097	1	2	-4836%
sqr6	197	3	3	8	200	1	1	-2%
bench	198	3	3	4	256	1	2	-29%
in5	446	7	7	4	1,17E+08	6	6	-26331853%
newtpla1	33	5	5	4	1024	2	3	-3003%
m3	571	4	4	16	1024	1	1	-79%
newapla	83	6	6	4	9728	3	3	-11620%
l8err	454	4	4	16	512	1	1	-13%
t4	238	6	6	16	8192	1	2	-3342%
br1	252	6	6	16	8192	1	2	-3151%
fout	442	3	3	4	320	1	2	28%
mp2d	136	7	7	3	7	3	4	95%
alu2	225	5	5	3	1282	2	3	-470%
t3	131	6	6	3	4	3	3	97%
p3	385	4	4	3	1408	2	2	-266%

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	DOMINIO MULTI	COSTO MULTIVALORE	INPUT ALICE	INPUT BOB	% GUADAGNO
m1	125	3	3	3	225	1	2	-80%
bcd_div	54	2	2	3	32	1	1	41%
alu3	139	5	5	3	4	2	3	97%
in7	182	7	7	3	23085056	6	7	-12683997%
	393,87 COSTO BOOL MEDIO	4,87	4,87		4716541,87 COSTO MULTI MEDIO	1,93	2,43	

I costi vengono notevolmente abbassati nel caso del booleano mentre in quello multi valore rimangono più elevati. Analizzando Questo fenomeno potremmo trovare una spiegazione nel fatto che la sintesi sui circuiti binari è nettamente più sviluppata e sono stati scoperti metodi di sintesi più efficaci rispetto a quelli studiati per una logica multi valore.

Il tool MVSIS infatti, come affermato in precedenza, non è più stato sviluppato ne mantenuto. All'atto pratico si presenta infatti più lento nella sintesi ed ha una gestione della memoria che porta a dump di memoria durante l'analisi di circuiti di grandi dimensioni.

5.2 ABC

Avendo potuto appurare che, inizialmente il costo del circuito multi valore possiede dei costi più bassi per via del minor numero di input, ma che la logica binaria può appoggiarsi a degli strumenti di sintesi più ottimizzati, si è provato ad utilizzare un approccio ibrido tra i due.

Sfruttando quindi come input un circuito multi valore ad ABC, questo tool è in grado di convertire automaticamente il circuito da multi valore a binario, in modo tale da poter utilizzare i suoi algoritmi di sintesi nettamente più ottimizzati.

```
read_blif_mv ./prova/blfmv/amd.mv
strash
compress2
cl
resyn2
cl
write_blif_mv ./prova/blfmv/synth/abc/amd.mv
```

Rispetto a prima però vi sarà la differenza che l'output post sintesi non sarà più un circuito multi valore, bensì uno binario.

5.3 ABC e dominio fisso

Come con *MVSIS* è stato inizialmente utilizzato un dominio fisso per svolgere una prima analisi.

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	COSTO MULTIVALORE	% GUAD
amd	469	7	7	1860	-297%
tms	322	4	4	198	39%
pdc	2818	7	7	6032	-114%
mlp4	594	4	4	764	-29%
apla	343	5	5	213	38%
f51m	261	4	4	808	-210%
m4	974	4	4	643	34%
newtpla2	48	5	5	48	0%
test1	1095	4	4	354	68%
m2	339	4	4	320	6%
br2	179	6	6	110	39%
alu1	83	6	6	83	0%
sqr6	197	3	3	263	-34%
bench	198	3	3	47	76%
in5	446	7	7	1336	-200%
newtpla1	33	5	5	33	0%
m3	571	4	4	381	33%
newapla	83	6	6	81	2%
l8err	454	4	4	434	4%
t4	238	6	6	197	17%
br1	252	6	6	153	39%
fout	442	3	3	214	52%
mp2d	136	7	7	315	-132%
alu2	225	5	5	554	-146%
t3	131	6	6	160	-22%
p3	385	4	4	139	64%
m1	125	3	3	116	7%
bcd_div3	54	2	2	46	15%
alu3	139	5	5	406	-192%
in7	182	7	7	1676	-821%
	393,87	4,87	4,87	599,47	-0,55
	COSTO BOOL MEDIO			COSTO MULTI MEDIO	

E' ora possibile notare come i risultati siano già migliorati e, in molti casi, la sintesi del circuito multi valore risulti essere migliore di quella binaria, tranne in sporadici casi.

5.4 ABC e dominio variabile

Come ultimo test, come in MVSIS, abbiamo deciso di applicare anche l'approccio MCD.

NOME CIRC	COSTO BOOL	INPUT ALICE	INPUT BOB	COSTO MULTIVALORE
amd	469	7	7	1860
tms	322	4	4	184
pdc	2818	7	7	6032
mlp4	594	4	4	669
apla	343	5	5	213
f51m	261	4	4	344
m4	974	4	4	658
newtpla2	48	5	5	48
test1	1095	4	4	354
m2	339	4	4	350
br2	179	6	6	130
alu1	83	6	6	774
sqr6	197	3	3	234
bench	198	3	3	47
in5	446	7	7	1336
newtpla1	33	5	5	33
m3	571	4	4	388
newapla	83	6	6	81
l8err	454	4	4	338
t4	238	6	6	248
br1	252	6	6	136
fout	442	3	3	214
mp2d	136	7	7	1677
alu2	225	5	5	554
t3	131	6	6	354
p3	385	4	4	139
m1	125	3	3	112
bcd_div3	54	2	2	46
alu3	139	5	5	406
in7	182	7	7	1676
	393,87	4,87	4,87	654,50
	COSTO BOOL MEDIO			COSTO MULTI MEDIO

In questo caso possiamo appurare che tra i due approcci non ci sono differenze sostanziali, questo risultato potrebbe essere derivato dalle modalità con cui ABC legge i circuiti multi valore in input.

Chapter 6

Conclusioni

Una volta terminate tutte le analisi e le sperimentazioni eseguite sui circuiti binari e multi valore siamo stati in grado di trarre alcune informazioni molto importanti riguardanti essi e i loro funzionamenti. Tutti i dati ottenuti da questa sperimentazione ci hanno portati a notare come l'utilizzo della logica multi valore all'interno dei circuiti potrebbe rivelarsi molto interessante in termini di riduzione dei costi relativi ai valori di input che sono richiesti dai circuiti.

La limitazione principale che abbiamo potuto riscontrare è insita nella carenza di strumenti di sintesi a disposizione che possano essere considerati applicabili alla logica multi valore. Come anticipato, solamente **MVSIS** è in grado di gestire e interpretare questo tipo di logica, ma rimane un programma ora mai datato e non più mantenuto, di conseguenza non vi sono al suo interno tool sufficientemente aggiornati o meglio implementati. I nuovi strumenti messi a disposizione, come ad esempio **ABC**, accettano in ingresso circuiti multi valore ma, per poter fare questo, nel loro motore di sintesi essi vengono convertiti in circuiti booleani. L'efficienza e l'efficacia delle nuove funzioni di sintesi attuate da **ABC** hanno reso l'utilizzo di questo software la prima scelta da utilizzare nel caso di circuiti multi valore e binari.

Applicando un approccio ibrido siamo stati in grado di prendere il meglio dei sintetizzatori presentati. Sfruttando la riduzione degli input durante la conversione in un dominio maggiore di quello binario combinata all'alta efficacia degli strumenti di sintesi messi a disposizione di **ABC** abbiamo ottenuto i risultati migliori.

Circuito	**BINARIO**	**MULTIVALORE**	**MVSIS dom var**	**ABC dom var**	**Riduzione co**
amd	393216	196608	47568	1860	96,09%
tms	4096	1024	1024	184	82,03%
pdc	2621440	1310720	1310720	6032	99,54%
mlp4	2048	512	512	669	-30,66%
apla	12288	6144	1029	213	79,30%
f51m	2048	512	272	344	-26,47%
m4	4096	1024	1024	658	35,74%
newtpla2	4096	2048	2048	48	97,66%
test1	2560	1280	1280	354	72,34%
m2	4096	1024	1024	350	65,82%
br2	32768	8192	8192	130	98,41%
alu1	32768	8192	4097	774	81,11%
sqr6	705	256	200	234	-17,00%
bench	512	256	256	47	81,64%
in5	1835008	117440512	117440512	1336	100,00%
newtpla1	2048	1024	1024	33	96,78%
m3	4096	1024	1024	388	62,11%
newapla	40960	20480	9728	81	99,17%
l8err	1283	512	512	338	33,98%
t4	32768	8192	8192	248	96,97%
br1	32768	8192	8192	136	98,34%
fout	640	320	320	214	33,13%
MEDIA	230286,73	5409911,27	5402215,91	666,86	65,27%

Questa tabella rappresenta la differenza di costo percentuale tra i due metodi più efficaci provati per ogni motore di sintesi.

Possiamo notare come questo tipo di metodica, definita ibrida, risulti essere il migliore approccio testato durante questa sperimentazione. E' possibile dimostrarlo in quanto i risultati mostrano una riduzione dei costi computazionali in tutti i casi ad eccezione di due, i restanti risultati hanno una percentuale di guadagno sempre superiore al 30% e, nei casi di circuiti con grande quantità di nodi, anche superiore al 100%; riducendo notevolmente il costo computazionale.

La tecnica di analisi ibrida offre i risultati migliori in quanto:

- La conversione di dominio ibrida al minimo dimezza i valori di input da inserire dalle parti e l'algoritmo fa in modo che non venga selezionato un dominio eccessivamente grande limitando le possibilità a casi utili in contesti reali;
- **ABC** è un tool recente e sempre più ottimizzato, in continuo aggiornamento, e dispone di preset ottimizzati per la sintetizzazione dei circuiti.

La tabella riassume anche qual è stata l'evoluzione di questa analisi sperimentale: inizialmente ci si è concentrati su un metodo ottimale di conversione binaria e si è analizzato se effettivamente questa sperimentazione potesse avere dei reali benefici. Successivamente si è passati alla sintesi dei circuiti per capire se la sintesi di domini non binari potesse essere efficace come quella binaria. Una volta individuati i tool migliori si è cercato di capire quali tecniche di sintesi potessero essere le migliori, fino ad arrivare ai risultati finali con la constatazione che l'approccio ibrido, ad oggi, risulti la tecnica migliore da noi testata.

Lo studio dell'utilizzo della logica multi valore ha portato a dei risultati interessanti nonostante la limitazione data dalla scarsità e obsolescenza degli strumenti di sintesi. Le tecniche analizzate in un'ottica futura potranno affiancare le classiche metodologie binarie in quanto, analizzando i risultati, alcuni tipi di circuiti non giovano della conversione in multi valore generando costi maggiori rispetto alla controparte booleana.

Bibliography

[1] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 503–513, New York, NY, USA, 1990. Association for Computing Machinery. ISBN 0897913612. doi: 10.1145/100216.100287. URL https://doi.org/10.1145/100216.100287.

[2] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 578–590, 2016.

[3] Bertacco and Damiani. The disjunctive decomposition of logic functions. pages 78–82, 1997. doi: 10.1109/ICCAD.1997.643371.

[4] Robert K Brayton and Sunil P Khatri. Multi-valued logic synthesis. In *Proceedings Twelfth International Conference on VLSI Design.(Cat. No. PR00013)*, pages 196–205. IEEE, 1999.

[5] Donald Chai, Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Alan Mishchenko, and Robert Brayton. Mvsis 2.0 user's manual. *Department of Electrical Engineering and Computer Sciences*, 2003.

[6] Wenliang Du and Mikhail J Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22, 2001.

[7] Elena Dubrova. *Multiple-Valued Logic Synthesis and Optimization*, pages 89–114. Springer US, Boston, MA, 2002. ISBN 978-1-4615-0817-5. doi: 10.1007/978-1-4615-0817-5_4. URL https://doi.org/10.1007/978-1-4615-0817-5_4.

[8] M Gao, J-H Jiang, Y Jiang, Y Li, A Mishchenko, S Sinha, T Villa, and R Brayton. Optimization of multi-valued multi-level networks. In *Proceedings 32nd IEEE International Symposium on Multiple-Valued Logic*, pages 168–177. IEEE, 2002.

[9] Brett Hemenway, William Welser IV, and Dave Baiocchi. Achieving higher-fidelity conjunction analyses using cryptography to improve information sharing. Technical report, RAND PROJECT AIR FORCE SANTA MONICA CA, 2014.

[10] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[11] Yuji Kukimoto. Blif-mv. *The VIS Group, University California, Berkely*, 1996.

[12] A Programmer's Manual. Quick look under the hood of abc. 2006.

[13] Jie-Hong Jiang Minxi Gao. Mvsis. *The Notes of the International Workshop on Logic Synthesis*, 2001.

[14] Alan Mishchenko et al. Abc: A system for sequential synthesis and verification. *URL http://www. eecs. berkeley. edu/alanmi/abc*, 17, 2007.

[15] Peichen Pan. Continuous retiming: algorithms and applications. In *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, pages 116–121, 1997. doi: 10.1109/ICCD.1997.628857.

[16] Luís Miguel Silveira, Jacob K. White, Horácio C. Neto, and Luís M. Vidigal. On exponential fitting for circuit simulation. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 11(5):566–574, 1992. doi: 10.1109/43.127618. URL https://doi.org/10.1109/43.127618.

[17] Daniele Venturi. *Computazione a parti multiple*, pages 389–422. Springer Milan, Milano, 2012. ISBN 978-88-470-2481-6. doi: 10.1007/978-88-470-2481-6_14. URL https://doi.org/10.1007/978-88-470-2481-6_14.

[18] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991.

[19] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[20] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986. doi: 10.1109/SFCS.1986.25.