Curriculum Vitae et Studiorum di Marco Comini*

nato a BRESCIA il 21/02/1970

Indice

Il sottoscritto **Comini Marco**, nato a BRESCIA il 21/02/1970, cittadino dell'Unione Europea, dichiara, sotto la propria responsabilità, che quanto affermato nel seguito corrisponde a verità, ai sensi degli artt. 2 e 4 della L. 15/68, degli artt. 1 e 2 del D.P.R. 403/98 e degli artt. 46, 47 e 49 del D.P.R. 445/2000, consapevole ai sensi dell'art. 76 del D.P.R. 445/2000 che le dichiarazioni mendaci sono punite ai sensi del codice penale e delle leggi speciali in materia.

1 Dati Anagrafici

Residenza	via Tricesimo, 123 – 33100 Udine
Nazionalità	Italiano
Stato civile	Celibe
Servizio Militare	assolto con servizio civile sostitutivo in data 7/11/96
Home page	http://www.dimi.uniud.it/comini/

*Aggiornato il 21 maggio 2013

e-mail marco.comini@uniud.it

telefono Tel: +39.043.255.8447 – Fax: +39.043.255.8499

indirizzo Dipartimento di Matematica e Informatica
Università Di Udine
Via Delle Scienze, 206
33100 Udine
Italy

2 Studi, Posizioni Accademiche e Borse

1989

- **Diploma Perito Industriale** *specializzazione Informatica*, Istituto Tecnico Industriale Statale *"B.Castelli"*, Brescia con votazione 60/60.
- **Vincitore del concorso di ammissione** al Corso Ordinario Classe di Scienze Matematiche, Fisiche e Naturali della *Scuola Normale Superiore di Pisa*.

1989–93

- **Corso di Laurea in Scienze dell'Informazione** presso l'*Università di Pisa*.
- **Studente Corso Ordinario della *Scuola Normale Superiore di Pisa***, Classe di Scienze Matematiche, Fisiche e Naturali.

1993

- (5 mesi) **Borsa di studio per laureandi** (prolungata dopo la laurea) del *Consiglio Nazionale delle Ricerche*. (01/09/93–01/02/94)
- **Laurea in Scienze dell'Informazione**, *Università di Pisa* il 16/7/1993 con punti 110/110 e lode. Titolo della tesi: *Una Semantica Generalizzata per Programmi Logici Positivi*. Relatore Prof. G. Levi, contro-relatore Prof. F. Rossi.
- **Diploma di Licenza in Scienze dell'Informazione**, *Scuola Normale Superiore di Pisa*. (01/11/93)
- **Vincitore del concorso di ammissione** al Corso di Dottorato in Informatica presso il Dipartimento di Informatica dell'Università di Pisa.

1993–97 (4 anni) Studente del corso di **Dottorato di Ricerca** in Informatica presso il *Dipartimento di Informatica* dell'Università di Pisa. (01/11/93–01/11/97)

1997 **Presentazione della Tesi di Dottorato** al Collegio docenti del Dipartimento di Informatica di Pisa. Titolo della tesi: *An abstract interpretation framework for Semantics and Diagnosis of logic programs*. Relatore Prof. G. Levi; controrelatori internazionali: Prof. G. Ferrand, Prof. J. Maluszynski; contro-relatori interni: Prof. R. Barbuti, Prof. G. Gallo.

1998

- (6 mesi) **Borsa di studio per laureati** del *Consiglio Nazionale delle Ricerche*. (15/07/98–04/01/99)
- (7 mesi) **Ricercatore ospite** presso *Institutionen för Datavetenskap, Linköping Universitet, Sweden* (Dipartimento di Informatica, Università di Linköping, Svezia). (01/02/98–31/08/98)
- (4 mesi) **Borsa di Studio post-dottorato** presso *Institut National de Recherche en Informatique et en Automatique, Rocquencourt-Paris, France* (Istituto Nazionale della Ricerca in Informatica ed Automazione, Rocquencourt, Parigi, Francia). (01/10/98–31/01/99)
- **Discussione della Tesi di Dottorato** con esito favorevole in data 12/5/98. Commissione giudicatrice: Prof. A. De Santis, Prof. S. Martini e Prof. V. Ambriola.

1999-00

- (5 mesi) **Professore Visitatore** "Van Vleck Visiting Assistant Professor" presso *Department of mathematics, Wesleyan University, Middletown CT, USA* (Dipartimento di Matematica, Università di Wesleyan, Middletown, Connecticut, Stati Uniti). (01/01/99–31/05/99)

- (2 anni) **Borsa di studio post-dottorato** presso l'*Università degli studi di Pisa*, A.A. 1998-99 e 1999-00,

2000-05 Ricercatore presso l'Università di Udine (Presa di servizio in data 3/7/00, confermato dal 3/7/03)

2005-ora Professore Associato presso l'Università di Udine (Presa di servizio in data 02/11/05, confermato dal 02/11/08)

3 Lingue Straniere Conosciute

Inglese *Ottima* conoscenza della lingua inglese, specialmente il lessico tecnico. La preparazione consiste in

3 anni durante le scuole medie superiori;

5 anni durante il corso di diploma;

1 anno durante il primo anno alla Scuola Normale Superiore. Il corso è stato tenuto da un insegnante *madrelingua*;

6 mesi composti di tre corsi intensivi, a livello avanzato, di due mesi per la specializzazione della lingua, tenuti al Centro Linguistico Interdipartimentale di Pisa da insegnanti *madrelingua*.

La conoscenza della lingua si è **raffinata** durante alcuni brevi periodi di permanenza all'estero (USA, Israele, Svezia e Francia).

Spagnolo Conoscenza a *livello scolastico*, parlato *discreto*. La preparazione consiste in 1 anno di studio al Centro Linguistico dell'Ateneo di Udine. Il corso è stato tenuto da un insegnante *madrelingua*.

La conoscenza della lingua si è **raffinata** durante alcuni periodi di permanenza in Spagna.

Russo Conoscenza a *livello scolastico*. La preparazione consiste in 2 anni di studio durante i primi 2 anni alla Scuola Normale Superiore. Il corso è stato tenuto da un insegnante *madrelingua*.

4 Attività Scientifica

4.1 Il Gruppo di Ricerca diretto

Il mio lavoro di ricerca si inserisce all'interno delle attività del gruppo di ricerca FLIT (da me coordinato) che si occupa essenzialmente di sviluppare metodologie formali basate sulla semantica con cui realizzare strumenti (automatici) di supporto alla programmazione. In particolare le tematiche su cui ci concentriamo son le seguenti:

Analisi, Verifica e Correzione per Linguaggi Dichiarativi:

- Abstract Diagnosis, Abstract Verification e Analisi di Linguaggi Logico-Funzionali.
- Abstract Diagnosis, Abstract Verification e Analisi di Linguaggi Concorrenti Temporizzati.
- Abstract Diagnosis, Abstract Verification e Analisi di Linguaggi Logici.
- Type Inference di Linguaggi Funzionali, Logici e Logico-Funzionali.
- Verifica e correzione di programmi logico-funzionali.

Metodi Formali per il Web e l'Ingegneria del Software:

- Linguaggi di interrogazione e filtering approssimato di documenti semistrutturati.
- Metodi formali per l'analisi e verifica di siti Web.
- UML consistency: analisi di consistenza di diagrammi UML.
- UML quality: analisi di qualità diagrammi UML.

Membri del Gruppo

- Marco Comini (Associato)
- Demis Ballis (Ricercatore)
- Andrea Baruzzo (Post-Doc)
- Giovanni Bacci (Post-Doc)
- Laura Titolo (Dottoranda)
- Luca Torella (Dottorando)

Collaborazioni Scientifiche del gruppo

Il gruppo collabora (stabilmente) con

- Moreno Falaschi, Michele Baggi. Dipartimento di Scienze Matematiche e Informatiche, Università degli Studi di Siena
- Maria Alpuente, Salvador Lucas, Alicia Villanueva, Santiago Escobar, Daniel Romero. Departamento de Sistemas Informàticos y Computaciòn, Universidad Politécnica de Valencia.
- Giorgio Levi, Roberta Gori. Dipartimento di Informatica, Università di Pisa.
- Ferruccio Damiani. Dipartimento di Informatica, Università di Torino.
- Makoto Tatsuta. National Institute of Informatics, Giappone.

4.2 Tematiche/Descrizione dell'Attività di Ricerca

Linee di Ricerca

- Semantica dei Linguaggi Dichiarativi.
- Interpretazione Astratta.
- Diagnosi (Dichiarativa) Astratta di Programmi Logici.
- Verifica Astratta di Programmi Logici.
- Verifica/Diagnosi Astratta di Programmi Funzionali.
- Verifica/Diagnosi Astratta di Programmi Logico-Funzionali.
- Verifica/Diagnosi Astratta di Programmi Concorrenti Temporizzati.
- Sintesi Automatica di Specifiche per Linguaggi Dichiarativi
- Type Inference di Linguaggi Funzionali, Logici e Logico-Funzionali.
- Verifica/Diagnosi tramite Semantiche Categoriali.
- Verifica di Diagrammi UML con vincoli OCL.
- Analisi di qualità di Diagrammi UML con vincoli OCL.

Motivazioni/Prefazione

I linguaggi di programmazione dichiarativi hanno semantiche denotazionali molto eleganti e semplici da capire. Purtroppo le tecniche di manipolazione di programmi basate sulla semantica (come analisi statica, debugging, verifica e trasformazione) hanno bisogno per essere corrette di semantiche dal sapore un po' più operazionale (senza però dover arrivare troppo vicino ad una semantica operazionale) .

In letteratura si possono trovare una miriadi di differenti semantiche *sviluppate ad-hoc* in grado di modellare differenti caratteristiche (proprietà astratte) della semantica operazionale di riferimento, incluse quelle sviluppate specificatamente per l'analisi statica. Con semantica di riferimento si intende quella semantica che modella il comportamento (reale) osservabile durante una computazione.

Una prima motivazione che ha guidato (e guida) il mio lavoro di ricerca è lo sviluppo di uno schema completo in grado di derivare *sistematicamente* una semantica corretta e fully abstract a partire dalla proprietà (astratta) a cui si è interessati. Con questo schema diventa possibile trattare formalmente problemi come la relazione fra semantica operazionale e denotazionale e ragionare sulle loro proprietà (come composizionalità, correttezza e grado di precisione), nonché comparare formalmente diverse semantiche.

Un esempio emblematico di tecnica di manipolazione bastata sulla semantica (concernente proprietà dichiarative) che potenzialmente trae grande vantaggio dall'uso di semantiche

un po' più operazionali è il debugging dichiarativo. La motivazione più rilevante del mio lavoro di ricerca è quello di *applicare*, mano a mano che vengono sviluppati, *i risultati dello schema teorico* per estendere e migliorare le tecniche di manipolazione (debugging, verifica, ecc) presenti in letteratura per poter lavorare con proprietà astratte qualsiasi e in particolare con quelle tipiche dell'analisi statica, definibili mediante domini finiti (o meglio Noetheriani, come le dipendenze ground), dove qualsiasi operazione (potenzialmente infinita) diventa finita.

Lo strumento su cui si basano tutte le mie costruzioni è l'Interpretazione Astratta.

L'Interpretazione Astratta in pillole

L'Interpretazione Astratta è una teoria di approssimazione di sistemi discreti ideata alla fine degli anni 1970 da P. e R. Cousot. Questa teoria permette di specificare formalmente processi di approssimazione, dimostrabilmente corretti, del comportamento di ogni sistema di calcolo. Per esempio, l'interpretazione astratta fornisce metodi che sono abbastanza generali da specificare analizzatori statici di programmi, verificatori automatici di sistemi software o hardware, verificatori automatici di proprietà di protocolli di comunicazione, sistemi di tipi e così via. Il campo di ricerca dell'interpretazione astratta è molto attivo: oltre alle conferenze dedicate al settore, l'*International Static Analysis Symposium* (SAS) e l'*International Conference on Verification, Model Checking and Abstract Interpretation* (VMCAI), molti contributi sono presenti anche nell'*International Conference on Computer Aided Verification* (CAV), *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (POPL) e altre conferenze/simposi. C'è una comunità scientifica molto significativa, specialmente in Europa (e in particolare in Italia, come testimoniato dai numerosi progetti PRIN su interpretazione astratta o tematiche affini), Stati Uniti e Asia.

Semantica dei Linguaggi Logici tramite Interpretazione Astratta [5,2,1,14,13,11] [1]

L'idea di usare le tecniche di interpretazione astratta come metodo unificante per le varie semantiche è ben nota [Cousot et al.]. L'originalità del mio lavoro comunque consiste nello sfruttare completamente questa idea e fornire realmente uno schema flessibile che fornisca solide e utili basi teoriche per *nuove applicazioni* basate sulla semantica.

La prima parte del lavoro è stata quella dello sviluppo di uno schema semantico per i programmi logici positivi adatto allo studio della correlazione fra le proprietà delle derivazioni SLD e tutte le loro astrazioni (proprietà astratte). Tale schema serve sia per la *ricostruzione* di semantiche esistenti sia (soprattutto) per lo sviluppo *sistematico* di *nuove* semantiche, in grado di trattare anche con l'*approssimazione* tipica dell'*analisi statica di programmi*.

Gli ingredienti del nostro schema sono una semantica concreta (che modella gli alberi *SLD*) e una proprietà astratta (astrazione di alberi *SLD*). La semantica denotazionale e il sistema di transizione (per la semantica operazionale) degli alberi *SLD* sono definiti in termini di quattro operatori semantici, direttamente correlati alla struttura sintattica del linguaggio. Ciò permette di ragionare sulle caratteristiche degli alberi attraverso una costruzione algebrica che da' anche una nuova comprensione riguardo lo spazio di ricerca delle soluzioni.

Grazie all'uso dell'interpretazione astratta per modellare la proprietà si è potuto formalizzare alcune condizioni algebriche rispetto ai quattro operatori semantici per classificare le proprietà e contemporaneamente garantire la validità di alcuni teoremi generali. A seconda della classe si ottiene automaticamente una nuova semantica denotazionale, un sistema di transizione, delle denotazioni top-down e bottom-up, assieme a vari interessanti teoremi (equivalenza delle semantiche, composizionalità delle stesse rispetto agli operatori sintattici, correttezza e minimalità delle denotazioni, loro grado di precisione). La tassonomia descrive, all'interno dello stesso framework, sia le semantiche usuali sia tutte le approssimazioni tipiche dell'analisi dei programmi. Inoltre, cosa molto importante, permette di ragionare in questo contesto sulle proprietà di composizionalità (condensing, OR-composizionalità, esistenza di un transition system astratto preciso).

[1] Per una spiegazione relativa al servizio CiteSeer.IST - Scientific Literature Digital Library si veda l'inizio della Sezione 4.5.

La nostra tassonomia è risultata essere molto utile per lo sviluppo di *nuove* semantiche aventi certe caratteristiche definite *a priori*. Infatti, dato che gli assiomi di ogni classe sono condizioni sufficienti affinché la semantica derivata goda di determinate caratteristiche, è sufficiente definire formalmente la proprietà astratta e verificare se appartiene alla giusta classe. Se non lo è, grazie alle operazioni di combinazione e raffinamento dell'interpretazione astratta, se ne può determinare una versione più concreta che sta nella classe desiderata.

I risultati di questa ricerca sono stati presentati in alcune conferenze internazionali [14,13,11]e in dettaglio nella tesi di dottorato [1], alcuni dei risultati sulla semantica di base sono illustrati in dettaglio in un lavoro pubblicato sulla rivista "Theoretical Computer Science" [2] ed alcuni dei risultati sullo schema complessivo sono illustrati in dettaglio in un lavoro pubblicato sulla rivista "Information and Computation" [5].

Diagnosi Astratta di Programmi Logici (con Vincoli) [3,33,1,17,32,16,15,30,12]

I risultati teorici descritti nella sezione precedente sono stati applicati con successo al campo del debugging (dichiarativo) di programmi logici.

Il debugging dichiarativo tratta essenzialmente con proprietà model-theoretic. La semantica di riferimento è il Modello Minimo di herbrand per [Shapiro, 82], l'insieme dei modelli di completamento per [Lloyd, 87] e l'insieme delle conseguenze atomiche per [Ferrand, 87]. L'idea alla base del Debugging Dichiarativo è raccogliere informazioni riguardo a quello che il programma dovrebbe fare e confrontarle con quello che esso realmente fa. Usando questi *sintomi* un programma di diagnosi trova poi gli errori.

Applicando la generalizzazione del nostro schema semantico sì è ottenuto un nuovo tipo di diagnosi dei programmi, chiamata "diagnosi astratta", che permette di considerare (in modo parametrico) una qualsiasi proprietà astratta (anziché la sola semantica denotazionale) e gode della flessibilità della tassonomia. In particolare è stata definita una metodologia di debugging, come semplice arricchimento della teoria base, mediante la quale rendere il debugging – nel caso di proprietà astratte che producono specifiche astratte finite – effettivo. Il guadagno di effettività va a scapito della precisione, ma sono stati mostrati alcuni semplici esempi di come all'atto pratico si ottengono importantissimi riscontri.

Partial diagnosis può essere utilizzata quando si ha a disposizione una conoscenza parziale (finita) del comportamento desiderato. Questa conoscenza può essere costruita tramite la rilevazione dei sintomi così come viene fatto in Diagnosi Dichiarativa. Questa formalizzazione può quindi essere vista come la formalizzazione teorica della Diagnosi Dichiarativa nel contesto astratto.

Diagnosis w.r.t. approximate observables Questa tecnica è utile quando di eseguono diagnosi rispetto a proprietà esprimibili come astrazioni su domini finiti (Noetheriani). Specifiche finite permettono la derivazione sistematica, a partire dalla teoria sottostante, di algoritmi *effettivi* di diagnosi che *non necessitano la rilevazione dei sintomi*.

Modular diagnosis Questa tecnica mostra che i metodi di diagnosi astratta non devono essere estesi per poter effettuare la diagnosi in modo modulare. Si può fare il debugging anche di programmi incompleti, basta solo avere la specifica delle componenti del programma mancanti.

Ho realizzato alcuni prototipi di strumenti di diagnosi implementando gli *algoritmi di diagnosi* su alcuni domini tipici. Il codice sorgente è disponibile pubblicamente sulla mia Home page.

Oltre ai risultati pratici ottenuti, la tecnica di Diagnosi Astratta risultante dall'applicazione dello schema semantico mostra che potenzialmente il nostro approccio è utile per definire molte nuove tecniche di manipolazione basate sulla semantica, utili per costruire potenti ed eleganti strumenti di supporto alla programmazione.

I risultati di tale ricerca sono stati presentati in alcune conferenze internazionali [17,16,15,12] e sono illustrati in dettaglio in un lavoro pubblicato sulla rivista "Journal of Logic Programming" [3], nonché nella tesi di dottorato [1].

Nell'ambito del progetto europeo Esprit DiSCiPL si è applicato lo stesso approccio della diagnosi astratta al caso più generale della Programmazione Logica con Vincoli (CLP). Alcuni primi risultati su questa linea di ricerca sono apparsi in un workshop internazionale [33] che poi è stata proseguita autonomamente dagli altri coautori dell'Università di Linköping.

Verifica di Programmi Logici (con Vincoli) [7,4,20,6,19,18]

Durante la ricerca relativa al debugging, illustrata nella sezione precedente, si è notato che fra la molte proprietà interessanti, ve ne erano alcune tipiche dei metodi per la verifica di proprietà di programmi. Una naturale conseguenza quindi è stata l'applicazione dello schema semantico al campo della verifica di programmi logici.

Lo scopo della verifica è la definizione di condizioni che permettano di provare formalmente che un programma si comporta correttamente rispetto alla specifica. In questo ambito si sono usate con successo le metodologie del framework semantico (e dell'interpretazione astratta) per l'organizzazione e la sintesi di metodi di verifica per programmi logici parametrici rispetto ad una proprietà astratta d'interesse (Abstract Verification Framework). Data una specifica proprietà le corrispondenti condizioni di verifica vengono derivate dal framework e sono garantite essere condizioni sufficienti per la correttezza parziale. Inoltre, scegliendo opportuni domini astratti che portino ad avere specifiche finite (Noetheriani), dette condizioni sono effettivamente calcolabili.

Ho realizzato, come caso di studio, un prototipo di strumento di verifica implementando gli *algoritmi di verifica* su un dominio di tipi [20]. Il codice sorgente è disponibile pubblicamente sulla mia Home page.

Il framework è in grado di ricostruire metodi ben noti, come la success-correctness [Clark79, Deransart93], la I/O correctness [Drabent97] e la I/O and call correctness [DrabentM88, BossiC89, AptM94].

Oltre che per studiare tecniche che consentano lo sviluppo sistematico di semantiche per modellare vari metodi di prova, il framework di verifica può essere istanziato anche al caso di specifiche fornite tramite un linguaggio d'asserzioni (che possono essere considerate come semantiche intensionali). Si sono infatti caratterizzati i linguaggi d'asserzioni come un dominio astratto e si sono studiati più in dettaglio due casi di studio (applicazioni) interessanti: un linguaggio di asserzioni decidibile e un'altro basato su clausole di Horn.

Il primo caso è un semplice linguaggio d'asserzioni decidibile, che è in grado di esprimere alcune proprietà base di termini, inclusi i loro tipi ed altre proprietà rilevanti per l'analisi statica. Stiamo attualmente studiando la definizione di altri linguaggi di specifica più espressivi (ma sempre decidibili).

Il secondo linguaggio permette all'utente di specificare alcuni predicati (da usare nelle asserzioni) attraverso programmi CLP. Si è visto come le condizioni di verifica derivate in questo caso possano essere provate applicando tecniche di trasformazioni di programmi ai predicati di specifica. Molte di queste condizioni si possono provare usando pochi passi di unfolding, mentre per provare condizioni più complesse si deve ricorrere a tecniche di trasformazione come il goal replacement. La generazione dei lemmi intermedi necessari per il goal replacement può essere ottenuta applicando metodi di prova unfold/fold come quelli di [PettorossiP99]. Attualmente sto studiando una combinazione degli algoritmi di verifica astratta con il tool MAP di [Pettorossi et al] per rendere il processo di prova semi-automatico.

I risultati di questa ricerca sono stati presentati in alcune conferenze internazionali [20,6,19,18] e sono illustrati in dettaglio in un lavoro pubblicato sulla rivista "Science of Computer Programming" [7].

Verifica/Diagnosi di Programmi Funzionali [45,9,21]

L'approccio alla Verifica, Diagnosi e Semantica dei Programmi Logici illustrato in precedenza può essere generalizzato ad altri paradigmi. Basta definire opportunamente una semantica concreta come punto fisso di un opportuno operatore. Recentemente si sono ottenuti promettenti risultati in tal senso per la programmazione funzionale.

Abbiamo ottenuto inizialmente in [21] un framework per il debugging astratto di programmi funzionali modellati come term rewriting systems. Si possono usare (a seconda delle

esigenze) due semantiche concrete per modellare le forme normali o i valori calcolati. Su queste semantiche si è costruito un framework di diagnosi e si è considerato in dettaglio un'istanza sul dominio $depth(k)$. Per questo caso abbiamo sviluppato un'implementazione in Haskell del diagnoser (DeBussy) e l'abbiamo utilizzata per verificare vari esempi non banali. Debussy è in grado di *verificare* la conformità di un programma OBJ (o un TRS) rispetto ad una specifica formale data intensionalmente mediante un altro programma OBJ (o TRS).

Purtroppo i risultati ottenuti, relativamente al tempo di esecuzione, non sono stati soddisfacenti quanto sperato. Il problema essenzialmente era dovuto all'uso di una semantica concreta goal-dependent, piuttosto di una semantica goal-independent, come invece fatto per il caso della programmazione logica.

Abbiamo quindi lavorato in [9] per definire una semantica compatta goal-independent per Term Rewriting Systems. Le semantiche tradizionali big-step delle forme normali e dei valori contengono molti elementi che sono in realtà conseguenza di un sottoinsieme di "comportamenti atomici" che caratterizzano tutti gli altri. La semantica di punto fisso proposta contiene solo gli elementi minimali che permettono di ricostruire, tramite chiusura per conseguenze di riscrittura, tutti gli altri elementi della semantica big-step. In effetti la semantica risultante è molto più piccola della big-step, arrivando anche ad essere finita per certi programmi (mentre la big-step non lo è mai). Per supportare quantitativamente questa affermazione è stato implementato un prototipo (disponibile all'URL `http://safe-tools.dsic.upv.es/zipit`) con cui son state misurate le dimensioni delle semantiche per vari programmi benchmark.

L'aspetto interessante della nuova metodologie ottenuta è che può essere applicata ad una classe di Term Rewriting Systems più ampia di approcci concorrenziali precedenti (come quelli di Echahed e Hanus).

I risultati di questa ricerca sono stati presentati in una conferenza internazionale [21] e in un articolo accettato per la pubblicazione presso la rivista "Theoretical Computer Science" [9].

Visto l'interesse mostrato dalla comunità scientifica per quanto fatto finora, intendo continuare in futuro a sviluppare a fondo il framework di verifica per i linguaggi funzionali.

45

Verifica/Diagnosi di Programmi Logico-Funzionali [43,26]

Come detto poc'anzi, l'approccio illustrato in precedenza per i linguaggi logici può essere generalizzato ad altri paradigmi. Un primo riscontro lo si è avuto appunto per i linguaggi funzionali. Attualmente stiamo lavorando anche all'estensione del framework per i linguaggi (integrati) logico-funzionali. Nel caso dei Linguaggi Logico-Funzionali la prima questione che deve essere risolta è la formalizzazione di una semantica dichiarativa goal-independent di punto fisso adatta a modellare le caratteristiche operazionali di interesse. Da questo punto di vista la letteratura non propone attualmente una soluzione denotazionale particolarmente soddisfacente neanche per il caso standard. Le sole proposte abbastanza studiate sono le versioni operazionali e quelle basate sulla logica della riscrittura, entrambe con formulazioni goal-dependent.

L'esperienza maturata negli anni passati ha dimostrato che gli approcci dell'Abstract Diagnosis e Abstract Verification risultano tanto più efficaci quando applicati ad una semantica concreta di punto fisso bottom-up composizionale che abbia una rappresentazione tanto più compatta. Quindi i nostri sforzi si devono naturalmente rivolgere nell'ottenere tale tipo di semantica concreta.

Data la mancanza in letteratura di precedenti candidati su cui investigare, abbiamo optato in [43] per l'utilizzo sistematico dell'Interpretazione Astratta:

1. Abbiamo definito, per la restrizione al prim'ordine del linguaggio Curry, una semantica (iper) concreta in grado di modellare completamente tutte le derivazioni di *needed narrowing*
2. Successivamente abbiamo definito l'astrazione delle risposte calcolate e derivato sistematicamente la sua semantica (che non è risultata precisa). Quindi abbiamo man mano raffinato detta astrazione fino ad ottenerne un raffinamento preciso.

Grazie a quest'ultima semantica abbiamo potuto applicare la metodologia dell'Abstract Debugging in [26] mostrando come si riescano a migliorare i precedenti approcci al debugging automatico per linguaggi logico-funzionali.

Attualmente si sta lavorando alla stesura di una implementazione di questa metodologia ed un articolo per generalizzare l'approccio al linguaggio completo (aggiungendo sia l'ordine superiore che le primitive rigide).

Verifica/Diagnosi di Programmi Logico-Funzionali [44,10, 47]

Attualmente stiamo lavorando anche all'estensione del framework per i linguaggi logici con vincoli concorrenti e temporizzati. Anche nel caso di questi linguaggi la prima questione che deve essere risolta è la formalizzazione di una semantica dichiarativa goal-independent di punto fisso adatta a modellare le caratteristiche operazionali di interesse.

Data la mancanza in letteratura di precedenti candidati su cui investigare, abbiamo iniziato a svilupparne una nostra in [44].

Grazie a quest'ultima semantica abbiamo potuto applicare la metodologia dell'Abstract Debugging in [10,47].

Sintesi Automatica di Specifiche per Linguaggi Dichiarativi [46,29,27,28]

Type-Inference via Abstract Interpretation [25]

Una istanza particolarmente interessante della metodologia di Verifica/Analisi Astratta si ha quando si sceglie come dominio astratto un dominio di Tipi. In questo caso la Verifica Astratta diventa il Type-Checking, mentre l'Analisi Astratta diventa il Type-Inference.

Dato che il dominio dei Tipi non è Noetheriano la convergenza al finito del calcolo del punto fisso astratto non può essere garantita. È noto che algoritmi di inferenza ad-hoc come quello di Damas-Milner, per garantire la terminazione in presenza di ricorsione polimorfa, utilizzano un passo di unificazione, che nel setting dell'Interpretazione Astratta corrisponde ad un operatore di widening particolarmente brutale. Ottenere quindi algoritmi di inferenza che riescono a tipare più programmi (garantendo sempre la terminazione) e abbastanza agevole in questo contesto perché basta semplicemente scegliere operatori di widening meno brutali. Tra l'altro il tipo si riesce ad inferire senza un eccessivo aggravio computazionale.

Il problema è che non si sa, in generale, quali siano i Type Systems che corrispondono a tali algoritmi di inferenza. Proprio di questo aspetto ci siamo occupati recentemente in un lavoro presentato in una conferenza internazionale [25] riuscendo tra l'altro a dare una caratterizzazione del Type System indecidibile di Milner-Mycroft mediante approssimazione con sistemi decidibili.

Implementazione di Linguaggi Logico-Funzionali

Le implementazioni esistenti per il linguaggio Curry (PAKCS e MCC) sono certamente meno mature delle implementazioni per il linguaggio Haskell (come GHC), principalmente data la recente introduzione dei linguaggi logico-funzionali rispetto a quelli funzionali.

Da un lato i linguaggi Haskell e Curry presentano caratteristiche sintattiche quasi identiche, anche se la semantica delle regole di definizione è notevolmente diversa. Dall'altro l'implementazione open-source GHC di Haskell gode di un numero ingente di librerie di supporto, che tra l'altro per loro natura sono inerentemente funzionali pure.

Recentemente abbiamo iniziato a riutilizzare i sorgenti di GHC per produrre una implementazione in grado di supportare codice multiparadigma Haskell/Curry, in modo da riuscire a mantenere tutte le librerie esistenti in GHC ma lasciando a disposizione l'espressività di Curry qualora lo si ritenga conveniente.

A tal scopo, tutta la prima parte della pipeline del compilatore è stata estesa in modo da poter trattare le estensioni di Curry, in special modo il TypeInferencer che deve andare a trattare i tipi primitivi di Curry relativi al non-determinismo. Inoltre, prendendo spunto dalla macchina astratta dell'implementazione di MCC (CAM), abbiamo iniziato ad ibridare la macchina astratta di GHC in modo da aggiungere le caratteristiche necessarie a supportare l'esecuzione di Curry, essenzialmente le variabili logiche e il non-determinismo, cercando di

mantenere le caratteristiche di esecuzione concorrente a thread di GHC che non esistono invece in Curry.

Lo sforzo per realizzare questa piattaforma è essenzialmente volto ad avere una implementazione reale in cui andare ad integrare tutte le metodologie di Analisi/Verifica che andremo man mano a sviluppare per i linguaggi Funzionali e Logico-Funzionali, in modo da poter avere una architettura implementativa comune (per poter così integrare vari domini astratti) e, soprattutto, un immediato riscontro pratico sulla bontà dei metodi proposti.

Verifica/Diagnosi tramite Semantiche Categoriali

Basandomi su alcuni risultati (molto preliminari) determinati durante la mia permanenza a Wesleyan con il Prof. Lipton intenderei sviluppare in futuro un framework semantico *categoriale* con cui riottenere i risultati illustrati in precedenza ed indagare sulle proprietà delle costruzioni canoniche della teoria delle categorie nel caso in esame. Ciò permetterebbe di estendere i risultati in modo parametrico ad altri paradigmi mediante l'immersione della categoria per il logic programming in una più generale, permettendo (presumibilmente) di arrivate a trattare anche i linguaggi imperativi.

Verifica/Qualità di Diagrammi UML con vincoli OCL [23,8,41,42,24,22]

Con l'avvento del Model-Driven Development i modelli son diventati una componente centrale del processo di sviluppo del software. Dato che dai modelli si può generare codice semi-automanticamente, la qualità del modello ha un impatto diretto sulla qualità del prodotto finale. Inoltre, la rilevazione di difetti a livello di modello ne permette l'eliminazione in fase iniziale quando è più facile e più economico.

Ci sono vari tools basati su metodi formali che possono essere utilizzati per attestare la qualità di un modello. Purtroppo questi metodi faticano ad essere utilizzati dalla comunità del software engineering essenzialmente a causa delle conoscenze matematiche che i professionisti del software devono acquisire per poter utilizzare detti strumenti.

Il nostro obiettivo, tramite l'ausilio dell'interpretazione astratta, è lo sviluppo di *metodi formali agili* che possano attestare la qualità di un modello tramite metodi automatici di analisi di modelli.

Abbiamo sviluppato 2 metodi complementari:

1. Verifica di Modelli per la Consistenza
2. Verifica di Modelli per la Qualità

Il primo metodo [24,41,22] è un primo passo verso l'obiettivo più ambizioso di ottenere Verifica di Modelli per la Correttezza, che serve per identificare inconsistenze fra diversi diagrammi (situazione abbastanza probabile a causa delle dimensioni tipiche dei modelli).

Il secondo metodo [23,8] invece serve per determinare automaticamente se in un modello sono (effettivamente) presenti istanze delle migliori pratiche di design del software (Design Patterns).

Lo spirito con cui son realizzate queste proposte è quello di poter aiutare lo sviluppatore a rilevare difetti di progettazione il prima possibile ma, soprattutto, nel modo meno invasivo possibile, utilizzando solo elementi standard di UML/OCL piuttosto che formalismi dedicati.

I metodi citati vengono utilizzati all'interno di una opportuna metodologia di sviluppo [42] che permette di integrarli con profitto in un tipico ambiente di Model-Driven Development.

I risultati di tale ricerca sono stati presentati in alcune conferenze internazionali [23,8,22].

4.3 Partecipazione a Progetti di Ricerca

Europei

1998 Membro del progetto ESPRIT (EU information technologies programme) "Debugging systems for constraint programming"

2004–2005 Membro del progetto EU "ICT for EU-India Cross-Cultural Dissemination" (grant ALA/95/23/2003/077-054)

Nazionali

1998–2000
- Membro del progetto Cofinanziato MURST "Sistemi formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software" (Formal Systems for the specification, analysis, verification, synthesis and transformation of software systems). [coordinatore G. Levi]
- Membro del progetto CNR "Verifica, analisi e trasformazione di programmi logici" (Verification, analysis and transformation of Logic Programs). [coordinatore G. Levi]

2000–2001 Membro del progetto Cofinanziato MURST "Certificazione Automatica di Programmi mediante Interpretazione Astratta" (Automatic program certification by abstract interpretation). [coordinatore R. Giacobazzi]

2001–2002 Membro del progetto Cofinanziato MURST "Interpretazione astratta, sistemi di tipo e analisi Control-Flow" (Abstract Interpretation, type systems and control-flow analysis). [coordinatore G. Levi]

2004–2005 Membro del progetto Cofinanziato MURST "Rappresentazione e gestione di dati spaziali e geografici in WEB" (WEB-based management and representation of spatial and geographic data) [coordinatore E. Bertino]

2005–2006 Membro del progetto Cofinanziato MURST "Interpretazione Astratta: Sviluppo e Applicazioni" (AIDA - Abstract Interpretation: Design and Applications) [coordinatore R. Giacobazzi]

2007–2008 Membro del progetto Cofinanziato MURST "Sistemi e calcoli di ispirazione biologica e loro applicazioni" (BISCA - Bio-Inspired Systems and Calculi with Applications) [coordinatore P. Degano]

Regionali

2002–2003 Membro del progetto Cofinanziato Regione FVG "Verifica formale, certificazione e model checking per sistemi reattivi, concorrenti ed embedded" (Formal verification, certification and Model-Checking for reactive, concurrent and embedded systems). [coordinatore A. Policriti]

4.4 Relatore di Tesi di Dottorato

2005-08 "A Unified Framework for Automated UML Model Analysis", Dipartimento di matematica e Informatica, Università di Udine (Andrea Baruzzo).

2009-12 "Abstract Diagnosis and Verification of Functional and Functional-Logic Programs", Dipartimento di matematica e Informatica, Università di Udine (Giovanni Bacci).

2010- "An Abstract Interpretation Framework for Semantics and Diagnosis of Term Rewriting Systems, Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena (Luca Torella).

2011- "An Abstract Interpretation Framework for Semantics and Verification of Timed Concurrent Constraint Languages, Dipartimento di matematica e Informatica, Università di Udine (Laura Titolo).

4.5 Elenco Pubblicazioni

Nel seguito si noti come per alcuni lavori viene riportato un numero etichettato CiteSeer. Questo numero è stato rilevato in data **10/03/05** tramite il servizio CiteSeer.IST - Scientific Literature Digital Library, servizio di riferimento internazionale per le citazioni in campo Informatico. Il numero riportato riferisce il numero di volte che l'articolo viene citato **esclusa le auto-citazioni**. Il servizio sembra non essere particolarmente aggiornato per quel che riguarda i lavori più recenti, ma certamente può fornire un parametro significativo per l'incidenza dei lavori con più di 5 anni.

Il servizio riporta inoltre un numero complessivo di citazioni **escluse le auto-citazioni** riguardanti un determinato autore che nel mio caso è **0**

Ho svolto un lavoro analogo con i servizi CiteSeer^X (ancora in beta) e Google Scholar riportando **0** e **0**.

Tesi di dottorato

1. M. Comini. *An abstract interpretation framework for Semantics and Diagnosis of logic programs.* Ph.D. thesis TD-5/98, Dipartimento di Informatica, Università di Pisa, 1998.

Riviste Internazionali (ISI Science Citation Index, con referee)

2. M. Comini and M. C. Meo. Compositionality properties of *SLD*-derivations. *Theoretical Computer Science*, 211(1-2):275–309, 1999. [2]
3. M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract diagnosis. *Journal of Logic Programming*, 39(1-3):43–93, 1999.
4. M. Comini, R. Gori, G. Levi, and P. Volpe. Abstract Interpretation based Verification of Logic Programs. *Electronic Notes in Theoretical Computer Science*, 30:1–17, 1999. [3]
5. M. Comini, G. Levi, and M. C. Meo. A Theory of Observables for Logic Programs. *Information and Computation*, 169:23–80, 2001.
6. M. Comini, R. Gori, and G. Levi. Logic programs as specifications in the inductive verification of logic programs. *Electronic Notes in Theoretical Computer Science*, 48:1–16, 2001.
7. M. Comini, R. Gori, G. Levi, and P. Volpe. Abstract Interpretation based Verification of Logic Programs. *Science of Computer Programming*, 49(1–3):89–123, 2003.
8. D. Ballis, A. Baruzzo, and M. Comini. A rule-based method to match Software Patterns against UML Models. *Electronic Notes in Theoretical Computer Science*, 219:51–66, 2007.
9. M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and J. Iborra. A Compact Fixpoint Semantics for Term Rewriting Systems. *Theoretical Computer Science*, 411(37):3348–3371, 2010.
10. M. Comini, L. Titolo, and A. Villanueva. Abstract Diagnosis for Timed Concurrent Constraint programs. *Theory and Practice of Logic Programming*, 11(4-5):487–502, 2011.

Atti di Conferenze/Workshop Internazionali (con almeno 3 referee)

11. M. Comini and G. Levi. An algebraic theory of observables. In M. Bruynooghe, editor, *Proceedings of the 1994 International Symposium on Logic Programming*, pages 172–186. The MIT Press, 1994. [4]
12. M. Comini, G. Levi, and G. Vitiello. Abstract debugging of logic programs. In L. Fribourg and F. Turini, editors, *Proceedings Logic Program Synthesis and Transformation*

[2] Già pubblicato come Technical Report 112, Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, 1996.

[3] Già pubblicato in versione preliminare in R. Echahed, editor, *Proceedings of the 8th International Workshop on Functional and Logic Programming*, pages 147–159, 1999.

[4] Già pubblicato come M. Alpuente, R. Barbuti, and I. Ramos, editors, *Proceedings GULP-PRODE'94*, pages 170–186, 1994.

and Meta-programming in Logic 1994, volume 883 of *Lecture Notes in Computer Science*, pages 440–450. Springer-Verlag, 1994.

13. M. Comini and G. Levi. Beyond the *s*-semantics: a theory of observables. In A. Ursini and P. Aglianò, editors, *Logic and Algebra*, volume 180 of *Lecture Notes in Pure and Applied Mathematics*, pages 25–67. Marcel Dekker, Incorporated, New York, 1995.
14. M. Comini, G. Levi, and M. C. Meo. Compositionality of *SLD*-derivations and their abstractions. In J. Lloyd, editor, *Proceedings of the 1995 International Symposium on Logic Programming*, pages 561–575. The MIT Press, 1995. [5]
15. M. Comini, G. Levi, and G. Vitiello. Declarative diagnosis revisited. In J. Lloyd, editor, *Proceedings of the 1995 International Symposium on Logic Programming*, pages 275–287. The MIT Press, 1995. [6]
16. M. Comini, G. Levi, and G. Vitiello. Efficient Detection of Incompleteness Errors in the Abstract Debugging of Logic Programs. In M. Ducassé, editor, *Proc. 2nd International Workshop on Automated and Algorithmic Debugging, AADEBUG'95*, pages 1–17, 1995.
17. M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Proving properties of logic programs by abstract diagnosis. In M. Dams, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, number 1192 in Lecture Notes in Computer Science, pages 22–50. Springer-Verlag, 1996.
18. M. Comini, R. Gori, and G. Levi. Assertion based Inductive Verification Methods for Logic Programs. In A. K. Seda, editor, *Proceedings of MFCSIT'2000*, volume 40 of *Electronic Notes in Theoretical Computer Science*, pages 1–18. Elsevier Science Publishers, 2001. [7]
19. M. Comini, R. Gori, and G. Levi. How to Transform an Analyzer into a Verifier. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings LPAR 2001*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 595–609. Springer-Verlag, 2001. [8]
20. M. Comini. VeriPolyTypes: a tool for Verification of Logic Programs with respect to Type Specifications. In M. Falaschi, editor, *Proceedings of 11th International Workshop on Functional and (constraint) Logic Programming*, number UDMI/18/2002/RR in Research Reports, pages 233–236, Udine, Italy, 2002. Dipartimento di Matematica e Informatica, Università di Udine.
21. M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas. Abstract Diagnosis of Functional Programs. In M. Leuschel, editor, *Logic Based Program Synthesis and Transformation – 12th International Workshop, LOPSTR 2002, Revised Selected Papers*, volume 2664 of *Lecture Notes in Computer Science*, pages 1–16, Berlin, 2003. Springer-Verlag. [9]
22. A. Baruzzo and M. Comini. Static Verification of UML Model Consistency. In D. Hearnden, J. G. Süß, B. Baudry, and N. Rapin, editors, *MoDeV²a: Model Development, Validation and Verification*. University of Queensland, Le Commissariat à l'Energie Atomique - CEA, October 2006.
23. D. Ballis, A. Baruzzo, and M. Comini. A Minimalist Visual Notation for Design Patterns and Antipatterns. In *5th International Conference on Information Technology: New Generations*, pages 51–56. IEEE Computer Society, 2008.
24. A. Baruzzo and M. Comini. A Methodology for UML Models V&V. In *Proceedings of First International Conference on Software Testing, Verification, and Validation*, pages 513–516. IEEE Computer Society, 2008.
25. M. Comini, F. Damiani, and S. Vrech. On Polymorphic Recursion, Type Systems, and Abstract Interpretation. In M. Alpuente and G. Vidal, editors, *Static Analysis – 15th International Symposium, SAS 2008*, volume 5079 of *Lecture Notes in Computer Science*, pages 144–158, Berlin, 2008. Springer-Verlag.
26. G. Bacci and M. Comini. Abstract Diagnosis of First Order Functional Logic Programs. In M. Alpuente, editor, *Logic-based Program Synthesis and Transformation,*

[5] Già pubblicato come M. I. Sessa, editor, *Proceedings GULP-PRODE'95*, pages 533–544, 1995.
[6] Già pubblicato come M. I. Sessa, editor, *Proceedings GULP-PRODE'95*, pages 607–618, 1995.
[7] Available at URL: http://www.elsevier.nl/locate/entcs/volume40.html
[8] Già pubblicato come *Joint International Conference on Declarative Programming – AGP 2001*, 2001.
[9] Già pubblicato come Technical Report DSIC-II/01/2003, Universidad Politécnica de Valencia, 2003

20th International Symposium, volume 6564 of *Lecture Notes in Computer Science*, pages 215–233, Berlin, 2011. Springer-Verlag.

27. G. Bacci, M. Comini, M. A. Feliú, and A. Villanueva. The additional difficulties for the automatic synthesis of specifications posed by logic features in functional-logic languages. In A. Dovier and V. S. Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, volume 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 144–153, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum Fuer Informatik.
28. G. Bacci, M. Comini, M. A. Feliú, and A. Villanueva. Automatic Synthesis of Specifications for First Order Curry Programs. In *Proceedings of the 14th symposium on Principles and practice of declarative programming*, pages 25–34, New York, NY, USA, 2012. ACM.
29. M. Comini and L. Torella. TRSynth: a Tool for Automatic Inference of Term Equivalence in Left-linear Term Rewriting Systems. In *ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation (PEPM'13)*. ACM, 2013. To appear.

Atti di Conferenze/Workshop Nazionali (con referee)

30. M. Comini, G. Levi, and G. Vitiello. On the Abstract Diagnosis of Logic Programs. In M.I. Sessa, editor, *Proceedings GULP-PRODE'95*, pages 41–57, 1995.
31. R. Bagnara, M. Comini, F. Scozzari, and E. Zaffanella. The AND-compositionality of CLP computed answer constraints. In M. Navarro, editor, *Proceedings of the APPIA-GULP-PRODE'96 Joint Conference on Declarative Programming*, pages 355–366, 1996.
32. M. Comini, G. Levi, and G. Vitiello. Modular abstract diagnosis. In J. L. Freire and M. Falaschi, editors, *Proceedings of the APPIA-GULP-PRODE'98 Joint Conference on Declarative Programming*, pages 409–420, 1998. [10]
33. M. Comini, W. Drabent, and P. Pietrzak. Diagnosis of CHIP programs using type information. In M. C. Meo and M. Vilares Ferro, editors, *Appia-Gulp-Prode'99, Joint Conference on Declarative Programming*, pages 337–349, L'Aquila, Italy, 1999. [11]
34. A. Baruzzo and M. Comini. Checking UML Model Consistency. In *Proceedings of CILC 2006 - Convegno Italiano di Logica Computazionale*, 2006.
35. M. Comini and G. Levi. An Algebraic Theory of Observables. In M. Alpuente, R. Barbuti, and I. Ramos, editors, *GULP-PRODE'94, Proceedings of the 1994 Joint Conference on Declarative Programming*, pages 170–186. Universidad Politécnica de Valencia, 1994.
36. G. Puebla, M. Comini, W. Drabent, M. Ducassé, M. Fabris, and C Schulte. Tools and Environments for (Constraint) Logic Programming (Workshop Overview). In J. Maluszynski, editor, *Logic Programming, Proceedings of the 1997 International Symposium*, pages 417–418, Cambridge, Mass., 1997. The MIT Press.
37. M. Comini, R. Gori, and G. Levi. Horn Clause Logic as specification language for program verification. 2001.
38. M. Comini and M. Falaschi. Electronic Notes in Theoretical Computer Science: Preface Volume 76. In M. Comini and M. Falaschi, editors, *Selected Papers of 11th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2002)*, volume 76 of *Electronic Notes in Theoretical Computer Science*, pages 1–2, North Holland, 2002. Elsevier Science Publishers. URL: http://www.sciencedirect.com/science/journal/15710661/76, doi:10.1016/S1571-0661(05)80798-4.
39. G. Bacci, M. Comini, M. A. Feliú, and A. Villanueva. Automatic Synthesis of Specifications for Curry Programs. In *Logic-based Program Synthesis and Transformation, 21th International Symposium pre-proceedings*, 2011. Accepted extended abstract.

[10] Già pubblicato in versione preliminare in *International Workshop on Tools and Environments for (Constraint) Logic Programming, ILPS'97 Postconference Workshop*, 1997.

[11] Già pubblicato in versione preliminare in *proceedings of Types for Constraint Logic Programming, postconference workshop of JICSLP'98*, 1998.

Rapporti Tecnici (senza referee)

40. M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. Technical Report 114, Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, 1996.

Lavori Sottomessi per Pubblicazione

41. A. Baruzzo and M. Comini. A Framework for Computer Aided Consistency Verification of UML Models. [12]
42. A. Baruzzo and M. Comini. Toward a Unified Framework for Quality and Consistency Verification of UML Models. [13]
43. G. Bacci and M. Comini. A Fully-Abstract Condensed Goal-Independent Bottom-Up Fixpoint Modeling of the Behaviour of First Order Curry. Technical Report DIMI-UD/06/2010/RR, Dipartimento di Matematica e Informatica, Università di Udine, 2010. [14].
44. M. Comini, L. Titolo, and A. Villanueva. A Condensed Goal-Independent Bottom-Up Fixpoint Modeling the Behavior of tccp. Technical report, DSIC, Universitat Politècnica de València, 2013. URL: http://riunet.upv.es/handle/10251/8351.
45. M. Comini and L. Torella. A Condensed Goal-Independent Fixpoint Semantics Modeling the Small-Step Behavior of Rewriting. Technical Report DIMI-UD/01/2013/RR, Dipartimento di Matematica e Informatica, Università di Udine, 2013. URL: http://www.dimi.uniud.it/comini/Papers/.
46. M. Comini and L. Torella. Automatic Inference of Term Equivalence in Term Rewriting Systems. Technical Report DIMI-UD/02/2013/RR, Dipartimento di Matematica e Informatica, Università di Udine, 2013. URL: http://www.dimi.uniud.it/comini/Papers/.
47. M. Comini, L. Titolo, and A. Villanueva. Abstract Diagnosis for tccp using a Linear Temporal Logic. URL: http://www.dimi.uniud.it/comini/Papers/.

Tesi di Dottorato di cui sono stato revisore

48. A. Baruzzo. *A Unified Framework for Automated UML Model Analysis*. PhD thesis, Dipartimento di matematica e Informatica, 2008.
49. G. Bacci. *An Abstract Interpretation Framework for Semantics and Diagnosis of Lazy Functional-Logic Languages*. PhD thesis, Dipartimento di matematica e Informatica, 2011.

4.6 Attività Scientifiche di Supervisione

2005 Responsabile Scientifico di un Assegno di Ricerca (Legge 449/97, art. 51) della durata annuale dal titolo: "Studio e sviluppo di un sistema di e-health per il controllo medico a distanza attraverso comunicazioni satellitari"

2006 Responsabile Scientifico di un Assegno di Ricerca (Legge 449/97, art. 51) della durata annuale dal titolo: "Studio e sviluppo di un sistema per teleconsulti medici a distanza attraverso comunicazioni multimediali"

2008 Membro della commissione internazionale per il conferimento del titolo di Dottore di Ricerca per la tesi di dottorato di Vicent Estruch dal titolo "Bridging the Gap between Distance and Generalisation: Symbolic Learning in Metric Space". Valencia, Dicembre 2008.

[12] Available at URL: http://www.dimi.uniud.it/comini/Papers/FramCAConVerUML/FramCAConVerUML.pdf
[13] Available at URL: http://www.dimi.uniud.it/comini/Papers/FrameworkQualConsUML/FrameworkQualConsUML.pdf
[14] Available at URL: http://www.dimi.uniud.it/comini/Papers/

4.7 Attività Scientifiche legate a Convegni e Workshop

1. **Chairman comitato di programma** International Workshop Tools and Environments for (Constraint) Logic Programming. Workshop post-conferenza International Symposium on Logic Programming, Port Jefferson, NY (USA), Ottobre 1997.
2. **AGP01 Membro comitato di programma** Joint International Conference Appia-Gulp-Prode'01, Evora (PT), Settembre 2001.
3. **WFLP02 Chairman comitato organizzatore** 11th International Workshop Functional and (Constraint) Logic Programming, Grado, Giugno 2002.
4. **WFLP02 Guest Editor** of *Selected Papers from WFLP'02 – 11th International Workshop on Functional and (Constraint) Logic Programming*, volume 76 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002. Available at URL: `http://www.elsevier.nl/locate/entcs/volume76.html`.
5. **WFLP08 Membro comitato di programma** 17th Int'l Workshop on Functional and (Constraint) Logic Programming, Siena, Luglio, 2008.
6. **VALID09 Membro comitato di programma** First International Conference on Advances in System Testing and Validation Lifecycle, Porto (Portugal), Settembre, 2009.
7. **LOPSTR10 Membro comitato di programma** 20th International Symposium on Logic-Based Program Synthesis and Transformation, Hagenberg (Austria), Luglio, 2010.
8. **VALID10 Membro comitato di programma** Second International Conference on Advances in System Testing and Validation Lifecycle, Nice (France), Agosto, 2010.
9. **TAP11 Membro comitato di programma** 5th International Conference on Tests and Proofs, Zurich (Switzerland), Giugno, 2011.
10. **TAP12 Membro comitato di programma** 6th International Conference on Tests and Proofs, Prague (Czech Republic), Maggio, 2012.
11. **TAP13 Membro comitato di programma** 7th International Conference on Tests and Proofs, Budapest (Hungary), Giugno, 2013.

4.8 Seminari su invito

1. Una caratterizzazione algebrica degli Osservabili. *Dipartimento di Informatica, Università di Salerno*, 20/1/95.
2. An abstract interpretation framework for semantics of logic programs. *IDA, Linköping Universitet, Sweden*, 24/4/98.
3. Abstract Diagnosis, an abstract interpretation framework for diagnosis of logic programs. *IDA, Linköping Universitet, Sweden*, 20/5/98.
4. The approach of Abstract Diagnosis and its recent extensions to type diagnosis of CHIP programs. *Rocquencourt research unit, INRIA, France*, 23/6/98.
5. (Ciclo di 5 seminari) Abstract Diagnosis. *LIFO project, Université d'Orleans, France*, Novembre'98-Gennaio'99.
6. An abstract interpretation framework for semantics of logic programs. *Sèminaire "Semantique et Interpretation Abstraite", Ecole Normale Superieure, Paris, France*, 20/11/98.
7. An abstract interpretation framework for abstract diagnosis of logic programs. *Sèminaire "Semantique et Interpretation Abstraite", Ecole Normale Superieure, Paris, France*, 27/11/98.
8. Abstract Diagnosis. *Université d'Orleans, France*, 14/12/98.
9. Abstract Interpretation based Verification and Diagnosis of Logic Programs. *Universidad Politecnica de Valencia, Spain*, 15/05/02.
10. Applications of Abstract Verification. *Universidad Politecnica de Valencia, Spain*, 22/05/02.
11. A Static Approach to Consistency Verification of UML Models. *Universidad Politecnica de Valencia, Spain*, 01/06/06.

4.9 Attività di Revisore

Riviste

1. TCS (2008,2010)
2. TPLP (2000-2001,2003,2004)

Convegni

1. APPIA-GULP-PRODE'96, AGP'01, AGP'02
2. ESOP'99
3. FLOPS'02
4. ICLP'99, ICLP'02, ICLP'03, ICLP'08
5. ICTCS'01, ICTCS'07
6. LICS'02
7. LOPSTR'10
8. LPAR'06
9. PEPM'08
10. PLILP/ALP'98
11. PPDP'03, PPDP'04, PPDP'08
12. SAC'05
13. TAP'11, TAP'12, TAP'13
14. TECLP'97
15. VALID'09, VALID'10
16. WFLP'01, WFLP'02, WFLP'07, WFLP'08
17. WWV'05

4.10 Chairman di sessioni a Convegni

1. Joint International Conference Appia-Gulp-Prode'00, La Havana (Cuba), Dicembre 2000.
2. International Workshop Functional and (Constraint) Logic Programming, Grado, Giugno 2002.
3. Joint International Conference Appia-GULP-PRODE 2002, Madrid (Spagna), Settembre 2002.
4. 17th Int'l Workshop on Functional and (Constraint) Logic Programming, Siena, Luglio, 2008.

4.11 Comunicazioni a Convegni

1. "A Generalized Semantic Framework for CLP", Workshop progetto nazionale "Modelli della Computazione e dei Linguaggi di Programmazione", Volterra, Settembre 1993.
2. "An Algebraic Theory of Observables", International Symposium on Logic Programming, Ithaca (USA), Novembre 1994.
3. "Compositionality of *SLD*-derivations", International workshop Abstract Interpretation of Logic Languages, Eilat (Israele), Giugno 1995.
4. "Compositionality of *SLD*-derivations and their abstractions", GULP-PRODE'95, Vietri, Settembre 1995.
5. "Approximated Abstract Compositional Semantics", International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 I), Pisa, Febbraio 1996.
6. "Proving Properties of Logic Programs by Abstract Diagnosis", International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 II), Gerusalemme (Israele), Dicembre 1996.
7. "Diagnosis of CHIP programs using type information", International workshop Types for Constraint Logic Programming, Manchester (UK), Giugno 1998.
8. "Diagnosis of CHIP programs using type information", Joint International Conference Appia-Gulp-Prode'99, L'Aquila, Settembre 1999.

9. "Design of Verification Methods by Abstract Interpretation", Workshop Finale del Progetto Cofinanziato "Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software." Venezia, Gennaio 2000.
10. "Logic programs as specifications in the inductive verification of Logic Programs", Joint International Conference Appia-Gulp-Prode'00, La Havana (Cuba), Dicembre 2000.
11. "Abstract Interpretation based Verification Methods", Workshop Progetto Cofinanziato "Certificazione Automatica di Programmi mediante Interpretazione Astratta" Venezia, Febbraio 2001.
12. "How to transform an analyzer into a verifier", Joint International Conference Appia-Gulp-Prode'01, Evora (PT), Settembre 2001.
13. "VeriPolyTypes: a tool for Verification of Logic Programs w.r.t. Type Specifications", International Workshop Functional and (Constraint) Logic Programming WFLP'02, Grado, Giugno 2002.
14. "The perspective of the Udine's Research Group within the AIDA project", Aida Febbraio 2005, Verona.
15. "An Effective Fixpoint Semantics for General Term Rewriting Systems", Aida 2005, Pisa.
16. "A Static Approach to Consistency Verification of UML Models", Aida 2006, Venezia.
17. "A Static Approach to Consistency Verification of UML Models", CILC 2006, Bari.
18. "On Polymorphic Recursion, Type Systems, and Abstract Interpretation", SAS 2008, Valencia.

4.12 Partecipazione a Convegni

1. IV Convegno Nazionale di Informatica Teorica, L'Aquila, Ottobre 1992.
2. EQUADIFF 8: Czecho-Slovak Conference on Differential Equations and their Applications, Bratislava (Slovakia), Agosto 1993.
3. Workshop progetto nazionale "Modelli della Computazione e dei Linguaggi di Programmazione", Volterra, Settembre 1993.
4. International Conference on Logic Programming, S. Margherita Ligure, Luglio 1994.
5. International Symposium on Logic Programming, Ithaca (USA), Novembre 1994.
6. International workshop Abstract Interpretation of Logic Languages, Eilat (Israele), Giugno 1995.
7. GULP-PRODE'95, Vietri, Settembre 1995.
8. International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 I), Pisa, Febbraio 1996.
9. International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 II), Gerusalemme (Israele), Dicembre 1996.
10. APPIA-GULP-PRODE'97, Grado, Giugno 1997.
11. International Workshop on Abstract Interpretation of Logic Languages (WAILL'97), Grado, Giugno 1997.
12. Joint International Conference and Symposium on Logic Programming (JICSLP'98), Manchester (UK), Giugno 1998.
13. International workshop Types for Constraint Logic Programming, Manchester (UK), Giugno 1998.
14. Joint International Symposia SAS and PLILP/ALP'98, Pisa, Settembre 1998.
15. Joint International Conference Appia-Gulp-Prode'99, L'Aquila, Settembre 1999.
16. Workshop Finale del Progetto Cofinanziato "Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software." Venezia, Gennaio 2000.
17. Joint International Conference Appia-Gulp-Prode'00, La Havana (Cuba), Dicembre 2000.
18. Workshop Progetto Cofinanziato "Certificazione Automatica di Programmi mediante Interpretazione Astratta" Venezia, Febbraio 2001.
19. International Workshop Functional and (Constraint) Logic Programming, Kiel (D), Settembre 2001.
20. Joint International Conference Appia-Gulp-Prode'01, Evora (PT), Settembre 2001.

21. International Workshop Functional and (Constraint) Logic Programming WFLP'02, Grado, Giugno 2002.
22. Joint International Conference SAS-LOPSTR-AGP 2002, Madrid (Spain), Settembre 2002.
23. Workshop "Giornata GULP 2002". Bologna, Ottobre 2002.
24. Workshop Progetto Cofinanziato "COVER" Bologna, Febbraio 2003.
25. International Workshop Functional and (Constraint) Logic Programming WFLP'03, Valencia (S), Giugno 2003.
26. "Convegno Italiano di Logica Computazionale" CILC04, Parma, Giugno 2004.
27. Aida Febbraio 2005, Verona.
28. Aida 2005, Pisa.
29. Aida 2006, Venezia.
30. CILC 2006, Bari.
31. WFLP 2008, Siena.
32. LOPSTR 2008, Valencia.
33. SAS 2008, Valencia.
34. LOPSTR 2010, Hagenberg.

4.13 Partecipazioni a Scuole

1. Prima Scuola Nazionale dei Dottorati di Informatica delle Facoltà di Scienze, Pontignano, Ottobre 1995.

4.14 Attività Scientifiche Locali

1. A. Baruzzo, M. Comini, Seminari su UML e OCL, Università degli Studi di Udine, Italy, Giugno-Luglio 2006.
   M. Comini:
   (a) Introduzione ad OCL. 06/07/06.
   (b) Caratterisitiche di OCL. 14/07/06.
   (c) Invarianti, Collections, Pre/Post-condizioni. 20/07/06.

   A. Baruzzo:
   (a) Diagrammi di Struttura. 30/05/06.
   (b) Diagrammi di Sequenza. 13/06/06.
   (c) Diagrammi di Attività/Casi d'Uso. 20/06/06.
   (d) Diagrammi di Stato. 04/07/06.
   (e) Caso di Studio I. 11/07/06.
   (f) Caso di Studio II. 18/07/06.

5 Attività in Organi Istituzionali/Incarichi Organizzativi

1. Rappresentante Associati in Giunta di Dipartimento del Dipartimento di Matematica e Informatica di Udine dal Novembre 2007 ad oggi.
2. Componente del Consiglio di Classe della Scuola Superiore, dal Settembre 2007 ad oggi.
3. Coordinatore Borse SOCRATES/ERASMUS con *Universidad Politecnica de Valencia* dal 2005 ad oggi.
4. Coordinatore Borse SOCRATES/ERASMUS con *Universidad de Castilla la Mancha* dal 2005 ad oggi.
5. Membro designato (dal Consiglio di Corso di Laurea) della Commissione Piani di Studio dei Corsi di Laurea in Informatica della Facoltà di Scienze, Università di Udine dall'Ottobre 2004 ad oggi.

6. Membro Rappresentante designato (dalla Facoltà di Scienze) nella Commissione Permanente delle Facoltà dell'Università di Udine dal 2004 ad oggi.

7. Membro del Collegio Docenti del Dottorato in Informatica del Dipartimento di Matematica e Informatica di Udine dal Novembre 2005 ad oggi.

8. Responsabile d'Ateneo del Centro di Competenza Universitario del progetto EUCIP4U (coord. Facoltà di Scienze e Facoltà di Ingegneria), dal Febbraio 2006 ad oggi.

9. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2008.

10. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2007.

11. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2006.

12. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2005.

13. Commissario a Procedura di Valutazione Comparativa a un posto di Ricercatore INF/01, facoltà di Medicina e Chirurgia, Università di Torino, Ottobre-Dicembre 2005.

14. Rappresentante eletto dei ricercatori nel Consiglio di Facoltà di Scienze dall'Ottobre 2003 all'Ottobre 2005.

15. Rappresentante dei Ricercatori in Giunta di Dipartimento del Dipartimento di Matematica e Informatica di Udine dal 13 Novembre 2002 al 31 Ottobre 2005.

16. Segretario dell'associazione nazionale Gruppo Utenti Logic Programming (GULP) dal Giugno 2000 al Dicembre 2005.

17. Rappresentante della Facoltà di Scienze Matematiche Fisiche e Naturali nella Commissione Permanente delle Facoltà dell'Università di Udine dal Dicembre 2004.

18. Membro della commissione Gestione Spazi Dipartimentali del Dipartimento di Matematica e Informatica di Udine negli anni 2001-2003.

19. Creatore nell'anno 2000 e successivamente gestore del sito Web del Gruppo Utenti Logic Programming (`http://www.dimi.uniud.it/gulp/`)

20. Creatore nell'anno 2002 e successivamente gestore del sito Web dell'11th International Workshop on Functional and (constraint) Logic Programming (`http://www.dimi.uniud.it/~wflp2002/`)

21. Addetto negli anni 2001-2003 alla gestione e trattamento dei dati delle pubblicazioni scientifiche del Dipartimento di Matematica e Informatica nel database d'ateneo.

6 Progetti Implementativi

Controllo a microprocessore di un braccio meccanico. Il sistema basato su Z80 è in grado di acquisire informazioni simboliche sui movimenti spaziali della mano robotica tramite una porta seriale RS-232 e di sviluppare run-time la strategia migliore per pilotare i motori passo-passo del braccio.

Editore guidato dalla sintassi. Realizzato in linguaggio SSL, è in grado di guidare l'utente nella scrittura di codice Pascal al fine di garantire la correttezza sintattica.

Interprete per la negazione costruttiva. Realizzato in linguaggio Prolog, permette di implementare il modello teorico della negazione costruttiva tramite vincoli equazionali con negazione.

Applicativo Multimediale. Realizzato con l'ambiente DIRECTOR 5.0 su piattaforma Macintosh PowerPC per la mostra di *Scultura Lignea* tenutasi a Lucca, dicembre 95 – giugno 96. Permette di consultare le schede tecniche delle opere d'arte, nonché di vedere le fotografie relative agli stadi intermedi del restauro e la collocazione delle opere all'interno delle sale della Mostra.

Abstract diagnosis debugger. Un programma PROLOG in grado di *trovare gli errori* in un qualsiasi altro programma in PROLOG puro, chiedendo interattivamente informazioni all'utente sul comportamento che ci si aspetta del programma esaminato.

Questo è uno dei progetti principali della tesi di dottorato.

Type Verifier. Un programma PROLOG in grado di *verificare* la conformità di un qualsiasi altro programma in PROLOG puro rispetto ad una specifica formale dei tipi intesi del programma.

Questo è il risultato della continuazione dei progetti della tesi di dottorato.

ZipIt. http://safe-tools.dsic.upv.es/zipit Un programma Haskell in grado di calcolare una semantica compatta goal-independent per Term Rewriting Systems secondo quanto specificato in [9]. Il tool accetta TRS in due formati di input: il formato TPDB o il sottoinsieme equazionale del linguaggio Maude.

AbsSpec http://safe-tools.dsic.upv.es/absspec/ A tool to automatically infer specifications from Curry programs. It statically infers from the source code of a Curry program a specification which consists of a set of equations relating (nested) operation calls that have the same behavior. We propose a (white-box) semantic-based inference method which relies on the (fully-abstract condensed) semantics of [49,27,28] for achieving, to some extent, the correctness of the inferred specification.

TRSynth http://safe-tools.dsic.upv.es/trsynth/ A tool (based on [46,29]) to automatically infer specifications from TRS programs. It statically infers from the source code of a TRS program a specification which consists of a set of equations relating (nested) operation calls that have the same behavior.

7 Attività Didattica e Lavorativa

7.1 Relazione di Tesi

7.1.1 Tesi di Laurea Specialistica

A.A.02-03 "Diagnosi Astratta di Linguaggi Funzionali", Corso di Laurea in Scienze dell'Informazione (Matteo Salsilli).

A.A.05-06 "Una semantica bottom-up goal-independent per programmi tccp", Corso di Laurea Specialistica in Informatica (Stefano Pramparo).

A.A.06-07 "Investigazioni su sisitemi di tipi per linguaggi funzionali con ricorsione polimorfa", Corso di Laurea Specialistica in Informatica (Samuel Vrech).

A.A.06-07 "Filtering approssimato di documenti XML", Corso di Laurea Specialistica in Informatica (Michele Baggi).

A.A.06-07 "Presentazione ragionata dell'implementazione MCC di Curry", Corso di Laurea Specialistica in Informatica (Marco Girol).

A.A.07-08 "Diagnosi Astratta di Curry al prim'ordine", Corso di Laurea Specialistica in Informatica (Giovanni Bacci).

A.A.09-10 "Analisi statica di Sistemi Reattivi Bigrafici tramite Interpretazione Astratta", Corso di Laurea Specialistica in Informatica (Emanuele D'Osualdo).

7.1.2 Tesi di Laurea Triennale

A.A.03-04 (doppia) "Progettazione ed implementazione di nuovi domini astratti per la verifica di programmi", Corso di Laurea (triennale) in Informatica (Samuel Vrech e Stefano Pramparo).

A.A.04-05 "Progettazione ed implementazione di un dominio astratto di tipi per linguaggi dichiarativi", Corso di Laurea (triennale) in Informatica (Matteo Dri).

A.A.04-05 "Uno strumento di Debugging per Full Haskell", Corso di Laurea (triennale) in Informatica (Angelo De Falco).

A.A.05-06 "Un tool per la conversione semi-automatica di programmi Curry in Haskell e viceversa", Corso di Laurea (triennale) in Informatica (Mauro Jacopo).

A.A.06-07 "Le costruzioni categoriali dei meccanismi di Haskell", Corso di Laurea (triennale) in Informatica (Emanuele D'Osualdo).

A.A.06-07 (tripla) "Estensione di GHC per supportare Curry", Corso di Laurea (triennale) in Informatica (Dario Meloni, Gianluca Sant e Luca Torella).

A.A.06-07 "Implementazione distribuita del linguaggio Pascal", Corso di Laurea (triennale) in Informatica (Matteo Cicuttin).

A.A.07-08 "Semantica Algebrica di CLP(FD)", Corso di Laurea (triennale) in Informatica (Laura Titolo).

7.2 Corsi di Dottorato

2005 Abstract Interpretation and Applications to Program Verification, Corso di Dottorato in Informatica, Dipartimento di Matematica e Informatica, *Università di Udine.* [20h]

2006 Abstract Interpretation, Corso di Dottorato in Informatica, Dipartimento di Scienze Matematiche e Informatiche "R. Magari", *Università di Siena.* [20h]

7.3 Corsi di Laurea, Esercitazioni e Laboratori

A.A.97-98 Incarico didattico per svolgere le esercitazioni del Corso di Programmazione I nel Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Pisa.* [20h]

A.A.99-00 Incarico didattico per svolgere le esercitazioni del Corso di Programmazione I nel Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Pisa.* [20h]

A.A.00-01

1. Laboratorio di Architetture, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
2. Laboratorio di Basi di Dati, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
3. Introduzione all'interpretazione astratta (all'interno del corso di Linguaggi di Programmazione), Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]

A.A.01-02

1. Programmazione Assembler (all'interno del corso di Architetture), Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
2. Laboratorio di Architetture, Corso di Laurea in *Università di Udine.* [48h]
3. Introduzione all'interpretazione astratta (all'interno del corso di Linguaggi di Programmazione), Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]

4. Laboratorio di Linguaggi II (all'interno del corso di Linguaggi di Programmazione), Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [4h]

A.A.02-03

1. Programmazione Assembler (all'interno del corso di Architetture), Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
2. Laboratorio di Architetture, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
3. Esercitazioni e laboratorio del corso di Linguaggi di Programmazione I, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
4. Esercitazioni e laboratorio del corso di Linguaggi di Programmazione II, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]

A.A.03-04

1. Laboratorio di Architetture (doppio corso, A e B), Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [96h]
2. Esercitazioni e laboratorio del corso di Linguaggi di Programmazione I, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
3. Esercitazioni e laboratorio del corso di Linguaggi di Programmazione II, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
4. Fondamenti di Informatica (Parte 1 e 2), master I livello "sistemi informativi territoriali", *ENAIP - FVG, Università di Udine, Comune di Tolmezzo, Associazione degli Industriali - Tolmezzo, Agemont* [50h]

A.A.04-05

1. Tecniche Formali per l'Ingegneria del Software, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
2. Esercitazioni e laboratorio del corso di Linguaggi di Programmazione I, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
3. Esercitazioni e laboratorio del corso di Linguaggi di Programmazione II, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
4. Sistemi Operativi, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [64h]
5. Programmazione nel paradigma logico con vincoli, Scuola Superiore, *Università di Udine.* Reading Course [16h]

A.A.05-06

1. Tecniche Formali per l'Ingegneria del Software, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
2. Linguaggi di Programmazione I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
3. Linguaggi di Programmazione II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
4. Il Paradigma di Programmazione Logico-Funzionale, Scuola Superiore, *Università di Udine.* Reading Course [16h]

A.A.06-07

1. Tecniche Formali per l'Ingegneria del Software, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]

2. Linguaggi di Programmazione I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Linguaggi di Programmazione II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]

A.A.07-08

1. Interpretazione Astratta e Analisi Automatica del Software, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Linguaggi di Programmazione I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Linguaggi di Programmazione II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
4. Algoritmi e Programmazione in Paradigmi Dichiarativi, Scuola Superiore, *Università di Udine*. Reading Course [16h]
5. Semantica Algebrica di CLP, Scuola Superiore, *Università di Udine*. Reading Course [16h]

A.A.08-09

1. Interpretazione Astratta e Analisi Automatica del Software, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Linguaggi di Programmazione I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Linguaggi di Programmazione II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
4. Elementi di semantica e programmazione, Scuola Superiore, *Università di Udine*. [16h]
5. Algoritmi e Programmazione in Paradigmi Dichiarativi, Scuola Superiore, *Università di Udine*. Reading Course [16h]
6. Approfondimenti del corso di Elementi di Semantica, Scuola Superiore, *Università di Udine*. Reading Course [16h]

A.A.09-10

1. Linguaggi di Programmazione I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Linguaggi di Programmazione II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Compilatori, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
4. Algoritmi e Programmazione in Paradigmi Dichiarativi, Scuola Superiore, *Università di Udine*. Reading Course [16h]

A.A.10-11 → ora Ogni anno

1. Linguaggi di Programmazione e Compilatori I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [72h]
2. Linguaggi di Programmazione e Compilatori II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [72h]
3. Interpretazione Astratta e Verifica Automatica del Software, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]

7.4 Attività Lavorativa extra Università

1996

1. Realizzazione di un applicativo multimediale per la mostra di *Scultura Lignea* tenutasi a Lucca, dicembre 95 – giugno 96;
2. Attività di amministratore di rete e di system manager presso il centro di calcolo della *Soprintendenza per i Beni Architettonici, Artistici e Storici* di Pisa nell'anno 96.

8 Conoscenze e Esperienze Programmative

Conoscenze teoriche. Ottima conoscenza di tutti i principali modelli di programmazione (imperativi, funzionali e logici). Conoscenze di base sulla programmazione ad oggetti. Conoscenze molto approfondite dei paradigmi logici: puro, con vincoli (reali, a dominio finito e razionali) e concorrente.

Sistemi Operativi. Buona conoscenza dei sistemi operativi UNIX su SUN e Digital, VMS su Digital, Linux su PC. Buona conoscenza dei sistemi MS-DOS, Windows 3.1 e Windows 95 su PC. Ottima conoscenza dei sistemi MacOS su Macintosh.

Linguaggi Assembler. Conoscenza specifica del linguaggio assembler Motorola 68000, Intel 8086, Zilog Z80, MIPS. L'uso di questi linguaggi è stato intrapreso nel corso di diploma per vari progetti sperimentali (tra cui la realizzazione del controllo a microprocessore di un piccolo braccio meccanico) e si è raffinato successivamente negli anni.

Linguaggio BASIC. L'uso di questo linguaggio è stato intrapreso nel corso di diploma come primo esempio pratico di linguaggio di alto livello. Tale linguaggio è stato utilizzato per lo sviluppo di piccoli progetti sperimentali e in varie altre occasioni.

Linguaggio C (Borland C, Microsoft C, THINK C, GNU C). Anche l'uso di questo linguaggio è stato intrapreso nel corso di diploma come linguaggio di alto livello per lo sviluppo di vari progetti sperimentali (tra cui la realizzazione di un piccolo data-base relazionale) e si è raffinato successivamente negli anni.

Linguaggio C++. La conoscenza di questo linguaggio, intrapresa a livello superficiale nel corso di laurea, è stata approfondita successivamente nella preparazione del corso di Linguaggi 1.

Linguaggio Java. La conoscenza di questo linguaggio, intrapresa a livello superficiale nel corso di laurea, è stata approfondita successivamente nella preparazione del corso di Linguaggi 1.

Linguaggio Pascal (TurboPascal, THINK Pascal). La conoscenza di questo linguaggio, intrapresa a livello superficiale nel corso di diploma, è stata molto approfondita durante il corso di laurea.

Linguaggi Concorrenti (ECSP e microprogr.). Anche la conoscenza di questi linguaggi è stata intrapresa durante il corso di laurea per la realizzazione di alcuni progetti di interesse accademico.

Linguaggi Concorrenti (Concurrent Constraint). La conoscenza di questo linguaggio è stata intrapresa durante il corso di dottorato per lo sviluppo di alcuni progetti di interesse prevalentemente accademico.

Linguaggi Logici (Edinburgh Prolog, Sicstus Prolog, CHIP, CC). Durante il corso di laurea la scelta dell'indirizzo di studio della programmazione logica mi ha portato a conoscere ed utilizzare moltissimi linguaggi logici, tra cui principalmente PROLOG e vari linguaggi per Constraint Logic Programming.

Linguaggi Funzionali (CAML, Haskell). Durante gli ultimi anni l'attività di ricerca mi ha portato a conoscere meglio alcuni linguaggi funzionali fra cui CAML e Haskell. Haskell in particolare anche a livello di implementazione.

Linguaggi Logico-Funzionali (Curry). Durante gli ultimi anni l'attività di ricerca mi ha portato a conoscere il linguaggio logico-funzionale Curry, anche a livello di implementazione.

Ambienti Multimediali. La conoscenza dell'ambiente DIRECTOR 5.0, per la realizzazione di filmati interattivi, è stata intrapresa per realizzare un applicativo multimediale su piattaforma Macintosh PowerPC.

9 Esami sostenuti durante gli studi

Esami Corso di Dottorato

Teoria della Dimostrazione	**20**/20
Proof Theory and Logic Programming	**20**/20
Logica Matematica	**20**/20
Programmazione Logica	**20**/20
Scuola di Pontignano	**20**/20
Complessità Computazionale	**20**/20
Proof Theory of Concurrency	**20**/20
Verification of Logic and PROLOG programs	**20**/20
Constraint Programming	**20**/20
Interpretazione Astratta	**20**/20
Metalogica	**20**/20

Esami Corso di Diploma della Scuola Normale Superiore

Seminario Fisico Matematico I	**25**/30
Lettorato di Lingua Russa I	**27**/30
Lettorato di Lingua Inglese III	**28**/30
Seminario Fisico Matematico II	**27**/30
Lettorato di Lingua Russa II	**24**/30
Word Problems	**29**/30
Calcolo delle Probabilità	**30**/30
Analisi Superiore	**30**/30

Esami Corso di Laurea

Geometria (sem)	**30**/30
Teoria ed Applicazioni delle Macchine Calcolatrici	**30**/30
Teoria degli Algoritmi e della Calcolabilità	**27**/30
Algebra (sem)	**30**/30 e **lode**
Analisi Matematica I	**29**/30
Sistemi per l'Elaborazione dell'Informazione I	**30**/30 e **lode**
Ricerca Operativa e Gestione Aziendale	**28**/30
Fisica I	**30**/30
Calcolo delle Probabilità e Statistica (sem)	**30**/30
Calcolo Numerico (sem)	**25**/30
Analisi Matematica II	**27**/30
Metodi per il Trattamento dell'Informazione	**30**/30
Sistemi per l'Elaborazione dell'Informazione II	**29**/30
Linguaggi Speciali di Programmazione	**30**/30
Fisica II	**30**/30
Linguaggi Formali e Compilatori	**30**/30 e **lode**
Elaborazione dell'Informazione Non Numerica	**30**/30 e **lode**
Progetto di Sistemi Numerici	**26**/30

Il sottoscritto dichiara, inoltre, di essere informato ai sensi e per gli effetti di cui all'art. 13 del D.L. 196/2003, che i dati personali raccolti saranno trattati, anche con strumenti informatici,

nell'ambito del procedimento per il quale la presente dichiarazione viene resa e per l'eventuale procedimento di assunzione in servizio e relativo trattamento di carriera.

Data, 21 maggio 2013 Fatto, letto e sottoscritto

(Marco Comini)