MINISTERO DELL'UNIVERSITÀ E DELLA RICERCA
DIREZIONE GENERALE PER IL COORDINAMENTO E LO SVILUPPO DELLA RICERCA
PROGRAMMI DI RICERCA SCIENTIFICA DI RILEVANTE INTERESSE NAZIONALE
RICHIESTA DI COFINANZIAMENTO (DM n. 1175 del 18 settembre 2007)

PROGETTO DI RICERCA - MODELLO A
Anno 2007 - prot. 2007NPW748

1 - Titolo del Progetto di Ricerca

Testo italiano
AIDA2007 - Interpretazione Astratta: Progettazione e Applicazioni

Testo inglese
AIDA2007 - Abstract Interpretation Design and Applications

2 - Durata del Progetto di Ricerca

24 Mesi

3 - Area Scientifico-disciplinare

01: Scienze matematiche e informatiche 100%

4 - Settori scientifico-disciplinari interessati dal Progetto di Ricerca

INF/01 - Informatica

5 - Coordinatore Scientifico

RANZATO	*FRANCESCO*	
Professore Associato confermato	*16/08/1969*	*RNZFNC69M16G224V*
Università degli Studi di PADOVA		
Facoltà di SCIENZE MATEMATICHE FISICHE e NATURALI		
Dipartimento di MATEMATICA PURA E APPLICATA		
0498271369 *(Prefisso e telefono)*	*0498271444* *(Numero fax)*	*ranzato@math.unipd.it*

* il progetto partecipa alla riserva del 10% di cui al punto 7 dell'art. 3

6 - Curriculum scientifico

Testo italiano
Francesco Ranzato ha conseguito la laurea in Matematica con lode nel 1993 e il dottorato di ricerca in Informatica Matematica nel 1997, entrambi presso l'Università di Padova. Nel 1995 è stato visitatore presso il Laboratoire d'Informatique dell'Ecole Polytechnique di Parigi. Dal 1997 al 1998 ha usufruito di borse post-dottorato finanziate dal CNR e dell'Università di Padova. Dal 1999 al 2002 è stato ricercatore di Informatica presso l'Università di Padova. Dal 2002 è professore associato di Informatica presso l'Università di Padova. Nel dicembre 2006 ha avuto una posizione temporanea di direttore di ricerca del CNRS francese presso l'Ecole Polytechnique di Parigi. Gli interessi di ricerca di Francesco Ranzato includono l'interpretazione astratta, l'analisi statica dei programmi, la semantica dei linguaggi di programmazione, la verifica automatica di sistemi mediante model checking, le equivalenze comportamentali nelle algebre di processi e la teoria dei reticoli. In particolare, egli ha contribuito alla definizione di un framework standard e diffusamente utilizzato per la progettazione sistematica di domini in interpretazione astratta e per la loro trasformazione attraverso raffinamenti/simplificazioni per ottenere proprietà di ottimalità come la completezza. Inoltre, egli ha contribuito all'integrazione ed alla combinazione di tecniche e strumenti tipici del model checking e dell'interpretazione astratta. Francesco Ranzato è stato o è membro di comitati di programma di congressi internazionali (International Static Analysis Symposium (SAS), International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)) ed organizzatore di workshop internazionali (International Workshop on Emerging Applications of Abstract Interpretation) su temi di analisi statica ed interpretazione astratta. È stato relatore invitato in workshop internazionali e in vari istituti di ricerca nazionali ed esteri (in Europa e negli Stati Uniti) e docente di corsi di dottorato di ricerca sull'interpretazione astratta. È membro del collegio docenti del dottorato di ricerca in Informatica delle Università di Bologna e Padova ed è stato membro di commissioni di esame finale per dottorati di ricerca italiani e Ph.D. europei. Francesco Ranzato tiene o ha tenuto corsi di Fondamenti di Informatica, Programmazione, Programmazione Orientata agli Oggetti per la laurea triennale, corsi di Semantica dei Linguaggi di Programmazione e Verifica e Analisi Statica per la laurea specialistica e corsi di Interpretazione Astratta per il dottorato di ricerca. È autore di più di 35 pubblicazioni riguardanti le aree precedentemente citate in riviste e congressi internazionali con revisione. Per quanto concerne la gestione di progetti di ricerca, Francesco Ranzato è stato responsabile scientifico nazionale di un progetto MIUR-FIRB triennale finanziato con 200.000 Euro, responsabile scientifico dell'unità di ricerca di Padova dei progetti MIUR-PRIN 2002 e 2004 finanziati con circa 70.000 Euro, responsabile scientifico di vari assegni di ricerca finanziati dall'Università di Padova e responsabile scientifico di vari progetti di ricerca ex-60% del gruppo informatico dell'Università di Padova.

Testo inglese

Francesco Ranzato received in 1993 the Laurea degree cum laude and in 1997 the Ph.D. in Computer Science, both at the University of Padova, Italy. On 1995 he visited the Laboratoire d'Informatique of Ecole Polytechnique, Paris, France. From 1997 to 1998 he held post doctoral positions funded by CNR (Italian National Research Council) and University of Padova. From 1999 to 2002 he was assistant professor in Computer Science at the University of Padova. From 2002 he is an associate professor in Computer Science at the University of Padova. On December 2006 he held a visiting "Directeur de Recherche" position of French CNRS at Ecole Polytechnique, Paris, France. His research interests include abstract interpretation, static program analysis, semantics of programming languages, automatic verification by model checking, behavioural equivalences in process algebras, lattice theory. In particular, he contributed to define a standard and widely used framework for systematically designing domains in abstract interpretation and for transforming them through refinements/simplifications in order to obtain optimality properties like completeness. Also, he contributed to the integration and combination of techniques and tools of model checking and abstract interpretation. Francesco Ranzato has been or is member of program committees of international conferences (International Static Analysis Symposium (SAS), International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)) and organizer of international workshops (International Workshop on Emerging Applications of Abstract Interpretation) on static analysis and abstract interpretation. He has been invited speaker at international workshops and at international research institutes (in Europe and US) and teacher of graduate courses on abstract interpretation. He is member of the scientific council of the joint graduate Ph.D. programme in Computer Science of the Universities of Bologna and Padova and has been member of international Ph.D. committees across Europe. He teaches or has taught Basic Computer Science, Programming and Object Oriented Programming at undergraduate level and Formal Semantics, Introduction to Verification and Static Analysis at graduate level, and Verification and Static Analysis by Abstract Interpretation at PhD level. He is author of more than 35 publications on the aforementioned areas in refereed international journals and conferences. As far as project funding and management is concerned, he has been or is principal investigator of a number of research projects concerning abstract interpretation and model checking, that have been funded by MUR (Italian Minister of University and Research) under action FIRB (~ 200.000 Euro) and action PRIN (~ 70.000 Euro) and by University of Padova (~ 40.000 Euro). He has been or is scientific supervisor of a number of post-doc research grants funded by University of Padova (~ 40.000 Euro).

7 - Pubblicazioni scientifiche più significative del Coordinatore Scientifico

1. *RANZATO F., F. TAPPARO. (2007). A new efficient simulation equivalence algorithm. Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS'07). (pp. 171-180). ISBN/ISSN: 0-7695-2908-9. doi:10.1109/LICS.2007.8: IEEE Press.*
2. *RANZATO F., TAPPARO F. (2007). Generalized strong preservation by abstract interpretation. JOURNAL OF LOGIC AND COMPUTATION. vol. 17, pp. 157-197 ISSN: 0955-792X. doi:10.1093/logcom/exl035.*
3. *GIACOBAZZI R., RANZATO F. (2006). Incompleteness of states w.r.t. traces in model checking. INFORMATION AND COMPUTATION. vol. 204(3), pp. 376-407 ISSN: 0890-5401. doi:10.1016/j.ic.2006.01.001.*
4. *RANZATO F., TAPPARO F. (2006). Strong preservation of temporal fixpoint-based operators by abstract interpretation. Proc. 7th Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'06). (vol. 3855 Springer LNCS, pp. 332-347). ISBN/ISSN: 3-540-31139-4. doi:10.1007/11609773.*
5. *GIACOBAZZI R, RANZATO F., SCOZZARI F. (2005). Making abstract domains condensing. ACM TRANSACTIONS ON COMPUTATIONAL LOGIC. vol. 6(1), pp. 33-60 ISSN: 1529-3785.*
6. *RANZATO F., TAPPARO F. (2005). An abstract interpretation perspective on linear vs. branching time. Proceedings of the 3rd Asian Symposium on Programming Languages and Systems (APLAS'05). (vol. 3780 Springer LNCS, pp. 69-85). ISBN/ISSN: 3-540-29735-9. doi:10.1007/11575467.*
7. *RANZATO F., TAPPARO F. (2005). An abstract interpretation-based refinement algorithm for strong preservation. Proc. 11th Intern. Conf. on Tools and Algorithms for the Construc. and Analysis of Systems (TACAS05). (vol. 3440 Springer LNCS, pp. 140-156). ISBN/ISSN: 3-540-25333-5. doi:10.1007/b107194.*
8. *RANZATO F., TAPPARO F. (2004). Strong preservation as completeness in abstract interpretation. Proceedings of the 13th European Symposium on Programming (ESOP'04). (vol. 2986 Springer LNCS, pp. 18-32). ISBN/ISSN: 3-540-21313-9.*
9. *GIACOBAZZI R, RANZATO F. (2002). States vs. traces in model checking by abstract interpretation. Proceedings of the 9th International Static Analysis Symposium (SAS'02). (vol. 2477 Springer LNCS, pp. 461-476). ISBN/ISSN: 3-540-44235-9.*
10. *RANZATO F. (2002). Pseudocomplements of closure operators on posets. DISCRETE MATHEMATICS. vol. 248(1-3), pp. 143-155 ISSN: 0012-365X.*
11. *RANZATO F., TAPPARO F. (2002). Making abstract model checking strongly preserving. Proceedings of the 9th International Static Analysis Symposium (SAS'02). (vol. 2477 Springer LNCS, pp. 411-427). ISBN/ISSN: 3-540-44235-9.*
12. *RANZATO F. (2001). A counterexample to a result concerning closure operators. PORTUGALIAE MATHEMATICA. vol. 58(1), pp. 121-125 ISSN: 0032-5155.*
13. *RANZATO F. (2001). On the completeness of model checking. Proc. 10th European Symposium on Programming (ESOP'01). (vol. 2028 Springer LNCS, pp. 137-154). ISBN/ISSN: 3-540-41862-8.*
14. *GIACOBAZZI R., RANZATO F., SCOZZARI F. (2000). Making abstract interpretations complete. JOURNAL OF THE ACM. vol. 47(2), pp. 361-416.*
15. *G. FILE', RANZATO F. (1999). The powerset operator on abstract interpretations. THEORETICAL COMPUTER SCIENCE. vol. 216, pp. 77-111 ISSN: 0304-3975.*
16. *GIACOBAZZI R., RANZATO F. (1999). The reduced relative power operation on abstract domains. THEORETICAL COMPUTER SCIENCE. vol. 216(1-2), pp. 159-211 ISSN: 0304-3975.*
17. *RANZATO F. (1999). Closures on CPOs form complete lattices. INFORMATION AND COMPUTATION. vol. 152(2), pp. 236-249 ISSN: 0890-5401.*
18. *GIACOBAZZI R, RANZATO F. (1998). Some properties of complete congruence lattices. ALGEBRA UNIVERSALIS. vol. 40(2), pp. 189-200 ISSN: 0002-5240.*
19. *GIACOBAZZI R, RANZATO F., SCOZZARI F. (1998). Complete abstract interpretations made constructive. Proc. 23rd Internat. Symp. Math. Foundations of Computer Science. (vol. 1450 Springer LNCS, pp. 366-377). ISBN/ISSN: 3-540-64827-5.*
20. *GIACOBAZZI R., RANZATO F. (1998). Uniform closures: order-theoretically reconstructing logic program semantics and abstract domain refinements. INFORMATION AND COMPUTATION. vol. 145(2), pp. 153-190 ISSN: 0890-5401.*
21. *GIACOBAZZI R., RANZATO F., SCOZZARI F. (1998). Building complete abstract interpretations in a linear logic-based setting. Proceedings of the 5th International Static Analysis Symposium (SAS'98). (pp. 215-229). ISBN/ISSN: 3-540-65014-8. vol. 1503 Springer LNCS.*
22. *RANZATO F., GIACOBAZZI R. (1998). Optimal domains for disjunctive abstract interpretation. SCIENCE OF COMPUTER PROGRAMMING. vol. 32(1-3), pp. 177-210 ISSN: 0167-6423.*
23. *CORTESI A., FILE' G., GIACOBAZZI R., PALAMIDESSI C., RANZATO F. (1997). Complementation in abstract interpretation. ACM TRANSACTIONS ON PROGRAMMING LANGUAGES AND SYSTEMS. vol. 19(1), pp. 7-47 ISSN: 0164-0925.*
24. *GIACOBAZZI R., RANZATO F. (1997). Completeness in abstract interpretation: a domain perspective. Proc. 6th Int. Conf. Algebr. Methodology and Softw. Tech. (AMAST'97), Sydney, AU. (pp. 231-245). ISBN/ISSN: 3-540-63888-1. Springer LNCS 1349.*
25. *GIACOBAZZI R., RANZATO F. (1997). Refining and compressing abstract domains. Proc. 24th Inter. Colloquium on Automata, Languages, and Programming (ICALP'97). (pp. 771-781). ISBN/ISSN: 3-540-63165-8. Springer LNCS vol. 1256.*
26. *FILE' G., RANZATO F. (1996). Complementation of abstract domains made easy. Proceedings of the 1996 Joint Int. Conf. Symp. Logic Program., Bonn, DE. (pp. 348-362). ISBN/ISSN: 0-262-63173-3. The MIT Press.*
27. *G. FILE', R. GIACOBAZZI, RANZATO F. (1996). A unifying view of abstract domain design. ACM COMPUTING SURVEYS. vol. 28, pp.*

333-336 ISSN: 0360-0300.

28. *GIACOBAZZI R., PALAMIDESSI C., RANZATO F. (1996). Weak relative pseudo-complements of closure operators. ALGEBRA UNIVERSALIS. vol. 36(3), pp. 405-412 ISSN: 0002-5240.*
29. *GIACOBAZZI R., RANZATO F. (1996). Complementing logic program semantics. Proc. 5th Inter. Conf. on Algebraic and Logic Programming (ALP'96), Aachen, DE. (pp. 238-253). ISBN/ISSN: 3-540-61735-3. Springer LNCS vol. 1139.*
30. *GIACOBAZZI R., RANZATO F. (1996). Compositional optimization of disjunctive abstract interpretations. Proceedings of the 6th European Symposium on Programming (ESOP'96). (pp. 141-155). ISBN/ISSN: 3-540-61055-3. Springer LNCS vol. 1058.*

8 - Elenco delle Unità operative

Unità	Responsabile dell'Unità di Ricerca	Qualifica	Ente	Impegno
I	*RANZATO Francesco*	*Professore Associato confermato*	*Università degli Studi di PADOVA*	*42*
II	*GIACOBAZZI Roberto*	*Professore Ordinario*	*Università degli Studi di VERONA*	*86*
III	*BAGNARA Roberto*	*Professore Associato confermato*	*Università degli Studi di PARMA*	*42*

9 - Abstract del Progetto di Ricerca

Testo italiano

Le scienze e tecnologie informatiche devono affrontare la grande sfida di sviluppare metodologie, tecnologie e strumenti adatti alla specifica dei requisiti, alla progettazione di architetture, all'implementazione, alla verifica e al test, all'utilizzo, al mantenimento e alla protezione sia durante il funzionamento che nell'evoluzione di software che sia componibile, affidabile, flessibile, scalabile e robusto. Avanzamenti ed innovazioni scientifiche sono necessari per mantenere e per guadagnare porzioni di mercato. L'astrazione, come formalizzata dall'interpretazione astratta, è indispensabile nei processi di sviluppo, integrazione, verifica, analisi, testing e protezione del codice. Lo scopo del progetto AIDA2007 -- che è la continuazione del progetto AIDA2004 che raggiunse pienamente gli obiettivi - è di rafforzare la ricerca italiana nell'ambito dell'interpretazione astratta, promuovendo l'interpretazione astratta come tecnologia per la costruzione di sistemi software affidabili. L'interpretazione astratta rappresenta la teoria di base per la progettazione di metodologie e per la costruzione di strumenti in grado si assicurare l'interoperabilità, la composizionalità, la scalabilità, l'affidabilità, la robustezza, la sicurezza e l'auto-adattamento del software. Si è dimostrato come l'interpretazione astratta sia una tecnologia fondamentale per software sequenziale, parallelo (sincrono e asincrono), a vincoli, distribuito e mobile che si trova in sistemi e servizi informatici, siano essi centralizzati, distribuiti o integrati. Recentemente, è stato inoltre dimostrato che l'interpretazione astratta può essere scalata ed applicata anche a complesse applicazioni industriali di grandi dimensioni. AIDA2007 si focalizzerà sullo sviluppo di metodologie fondazionali e su tre filoni di applicazioni: l'analisi statica dei programmi, la protezione del codice e la verifica dei sistemi. Nell'ambito delle metodologie fondazionali, considereremo la progettazione sistematica di domini ottimali per il model checking, la definizione di domini astratti per l'analisi di puntatori e di aliasing per programmi C e Java, la progettazione di domini astratti numerici che siano scalabili e precisi, lo sviluppo di metodi sistematici per misurare la precisione dell'analisi che integrano nei domini astratti dell'informazione probabilistica e statistica, e la definizione di trasformazioni semantiche che permettano di ottenere proprietà di completezza. Per quanto riguarda l'analisi statica di programmi, affronteremo i seguenti problemi: specifica ed analisi di proprietà causali in sistemi concorrenti e distribuiti, ottimizzazione speculativa, analisi di terminazione e complessità di programmi C e Java mediante interpretazione astratta, definizione di astrazioni numeriche modulari, estensione di linguaggi mediante annotazioni nelle fasi di sviluppo e analisi del codice, analisi precise di string cleanness. Nell'ambito della protezione del codice, studieremo il problema di nascondere informazione nei linguaggi di programmazione, con particolare attenzione al software watermarking, alla protezione del copyright, all'offuscamento di codice per impedire il reverse engineering e all'identificazione di malware in presenza di malware metamorfici (ovvero offuscati). Per quanto riguarda la verifica automatica di sistemi, studieremo algoritmi di raffinamento delle astrazioni per il model checking astratto, trasformazioni di linguaggi temporali per ottenere la preservazione forte rispetto ad una data astrazione, e algoritmi efficienti basati su interpretazione astratta per calcolare equivalenze comportamentali in algebre di processi.

Testo inglese

Information sciences and technologies face the great challenge of developing effective methodologies, technologies, and tools for the requirement specification, architecture design, implementation, verification and test, deployment, maintenance and protection in workings and evolutions of composable, reliable, flexible, scalable and robust software. Scientific breakthroughs and innovation are required to maintain and gain market shares. Abstraction, as formalized by abstract interpretation, is mandatory in the development, integration, verification, analysis, testing and protection processes. The aim of the AIDA2007 project -- which is the continuation of the past successful AIDA2004 project -- is to strengthen the Italian research in abstract interpretation, by promoting abstract interpretation as a basic enabling technology required to build dependable software systems. Abstract interpretation will serve as the foundational theory to design methodologies and to build tools to ensure the interoperability, composability, scalability, reliability, robustness, dependability, security and autonomous self-adaptation of software. This theory has been proved to be a key technology for sequential, constraint, parallel (synchronous or asynchronous), distributed and mobile software as found in software and services -- whether centralized, distributed or embedded -- and has been recently shown to scale up for large industrial applications. AIDA2007 will focus on both basic design methodologies and three key applications in: static program analysis, code protection and system verification. In the basic methodologies we will consider the systematic design of optimal domanis for model checking, the design of abstract domains for points-to and aliasing analysis of C and Java programs, the design of scalable and precise numeric abstract domains, the development of systematic methods for measuring the precision of analyses by improving domains with probabilistic and statistical information, and semantic transformations for achieving completeness. In the context of static program analysis, we will attack the problems of: causality in concurrent and distributed systems, speculative optimization, termination and complexity analysis by abstract interpretation of Java and C code, modularity in numerical abstractions, language extensions by annotation in code design and analysis, and precise analyses of string cleanness. In code protection we will attack the problem of information concealing in programming languages, in particular for software watermarking, copyright protection, code obfuscation against reverse engineering and malware detection in presence of metamorphic (obfuscated) malware. In the context of automatic system verification we will investigate abstraction refinement algorithms for abstract model checking, transformations of temporal languages for achieving strong preservation against a given abstraction, and efficient algorithms based on abstract interpretation for computing behavioural equivalences in process algebras.

10 - Parole chiave

n°	Parola chiave (in italiano)	Parola chiave (in inglese)
1.	*INTERPRETAZIONE ASTRATTA*	*ABSTRACT INTERPRETATION*
2.	*ANALISI STATICA*	*STATIC ANALYSIS*
3.	*DOMINI ASTRATTI*	*ABSTRACT DOMAINS*
4.	*VERIFICA DI SISTEMI*	*SYSTEM VERIFICATION*
5.	*SEMANTICA FORMALE*	*FORMAL SEMANTICS*

11 - Obiettivi finali che il Progetto si propone di raggiungere

Testo italiano

Lo straordinario sviluppo delle tecnologie hardware e software è all'origine della moderna società dell'informazione e della comunicazione. Promotrici della più grande rivoluzione economica e culturale del secolo scorso, tali tecnologie rappresentano tuttavia il tallone d'Achille della nostra società, perché sia in campo software che hardware requisiti quali affidabilità e sicurezza sono difficili da garantire. Ciò motiva la crescente importanza ed il costo assunti dalla validazione di software e hardware. Gli strumenti tradizionali di validazione hanno tuttavia una scalabilità ridotta e costi sempre maggiori, sono spesso inefficienti, vengono rilasciati lentamente e non garantiscono una qualità ottimale.

L'Interpretazione Astratta è un metodo generale, sia teorico che pratico, per approssimare la semantica del comportamento di un sistema di calcolo. Gli strumenti di validazione del software -- come analisi statica, model checking, type and effect system -- rivelano aspetti unificanti se riformulati in termini di interpretazione astratta. Tali strumenti descrivono in modo sicuro ma approssimato il comportamento di un sistema di calcolo astraendone sistematicamente la semantica, il modello o il programma. L'interpretazione astratta non solo formalizza queste astrazioni, ma ne permette pure il raffinamento e la combinazione, e si pone altresì come base per la modellazione di strumenti per la verifica di tali proprietà a vari livelli di astrazione. Nell'ultimo decennio essa è andata ben oltre i normali circoli accademici, mostrando un ingente potenziale come tecnologia d'avanguardia per le applicazioni industriali. Alcuni esempi: AbsInt (www.absint.com) commercializza aiT, un'applicazione basata su interpretazione astratta che fornisce precise approssimazioni del tempo d'esecuzione di programmi real-time eseguiti su processori di sistemi integrati impiegati nell'industria aeronautica e automobilistica; la componente Polyspace (www.polyspace.com) di MathWorks fornisce uno strumento semantico per l'analisi automatica di software integrato in Ada, C e C++ e il debugging di errori a run-time; l'analizzatore statico Astrée (www.astree.ens.fr) è in grado di verificare automaticamente l'assenza di errori run-time in programmi per il controllo di volo di alcuni aerei civili europei (come l'AIRBUS A380), programmi che sono tipicamente sincroni, real-time e safety-critical, e contano più di 500.000 linee di codice C. Un altro segno dei tempi è il GlobalGCC project (GGCC) nell'ambito dell'ITEA programme (Information Technology for European Advancement), il cui obiettivo è estendere la GNU Compiler Collection (GCC) con tecniche di analisi statica globale. Sebbene i tempi siano maturi, ci sono tuttavia ancora problemi rilevanti da risolvere prima che questa rivoluzione, legata allo sviluppo di software critico, possa estendersi su larga scala.

Lo scopo primario del progetto AIDA2007 è di sostenere la comunità italiana di ricerca sull'interpretazione astratta promuovendo tale metodologia come tecnologia fondamentale per lo sviluppo di software affidabile. Ci proponiamo di mobilizzare, unificare e galvanizzare i ricercatori italiani di interpretazione astratta, creando laboratori virtuali di ricerca pura e applicata guidati dai medesimi obiettivi scientifici. Questo si realizzerà integrando le competenze dei gruppi di ricerca sull'interpretazione astratta già esistenti in Italia, specialmente quelli di Padova, Parma e Verona, che ne sono espressione trainante. L'interpretazione astratta servirà da teoria fondazionale per la progettazione di metodi e strumenti che assicurino interoperabilità, composizionalità, scalabilità, affidabilità, robustezza, sicurezza e auto-adattività del software.

Per quanto riguarda la ricerca di base in interpretazione astratta, siamo interessati allo sviluppo di domini e semantiche astratte. È notorio che le proprietà di un'intepretazione astratta dipendono fortemente dalle proprietà della semantica sottostante e dal tipo di astrazione operata. Noi affronteremo questo problema fondamentale studiando le astrazioni e le semantiche in rapporto alle proprietà che l'interpretazione astratta risultante deve soddisfare. Ciò trova applicazione nella definizione sistematica di domini ottimali per il model checking, di domini astratti per l'analisi di puntatori e di aliasing in C e Java, e di domini astratti numerici precisi e scalabili, oltre che nello sviluppo di metodi sistematici per misurare la precisione di un'analisi mediante informazione probabilistica e statistica e di trasformazioni semantiche che inducano proprietà di completezza. L'aspetto trasversale a questi temi è che tutti si riducono a processi di raffinamento dei domini astratti. La teoria delle trasformazioni dei domini astratti offrirà un'infrastruttura comune tramite la quale arricchire lo stato dell'arte in questo campo con nuovi, efficienti e precisi domini astratti.

Per quanto riguarda le applicazioni, vogliamo esplorare nuove direzioni nell'analisi statica di programmi, nella protezione del codice e nella verifica di sistemi. Nell'analisi statica di programmi, perlustreremo nuove idee desunte dall'analisi di complessità e terminazione di codice C e Java, e dall'analisi di dipendenze causali. Studieremo tecniche per provare automaticamente la terminazione o per maggiorare la complessità di frammenti di programma sfruttando opportune informazioni di natura metrica nei domini astratti. Un tale approccio, basato sulla teoria dei domini, ci consentirà di modellare le dipendenze causali fra azioni di processi in sistemi concorrenti e distribuiti, soprattutto nel CCS e nel pi-calculus, in termini sia della loro architettura spaziale, sia delle loro capacità di comunicazione. Mentre l'analisi statica è un ambito ben definito in cui l'interpretazione astratta gioca un ruolo chiave da decenni, ciò non accade nella verifica di sistemi e nell'emergente settore della protezione del codice. Nella verifica di sistemi siamo interessati ad algoritmi per il raffinamento delle astrazioni, alla trasformazione di linguaggi temporali e ad algoritmi efficienti per calcolare equivalenze comportamentali. Intendiamo studiare come sia possibile applicare l'interpretazione astratta nel contesto dei raffinamenti di astrazione guidati da controesempi, e per progettare algoritmi più accurati per il raffinamento di astrazioni nel model checking astratto. Studieremo simmetricamente le trasformazioni di linguaggi temporali che permettano di eludere i controesempi spuri. Nell'ambito della protezione del codice, considereremo i rischi connessi agli attacchi perpetrati sia da host che da software maligni. Gli host maligni tipicamente ricercano nei programmi dati confidenziali, usano analisi statiche e dinamiche per estrarne informazioni, li decompongono per riutilizzarne il codice, ne corrompono l'integrità. I software maligni sfruttano le falle di altri programmi per inserirvi funzionalità maligne. Gli host maligni sono i maggiori responsabili della pirateria informatica. Contro di essa e contro la reverse engineering maligna, strumenti come software watermarking e offuscamento del codice si pongono come adeguate tecniche di difesa. Intendiamo delineare nuove e robuste tecniche di software watermarking e offuscamento del codice basate su interpretazione astratta. Queste tecniche, oltre ad ostacolare la reverse engineering e la violazione del copyright, offriranno una teoria fondazionale agli attacchi del software maligno. La minaccia rappresentata da questi software è una sfida attuale alla sicurezza dei sistemi informatici. Intendiamo pertanto introdurre, accanto all'attuale riconoscimento sintattico del software maligno, algoritmi di rilevazione basati sulla semantica e sulle sue proprietà.

Testo inglese

The great development of software and hardware technology is at the origin of the modern information and communication society. On the one hand these technologies have provided one of the greatest economical and cultural revolution in the last century, on the other hand they represent the Achille heel of our modern society, becuase software and hardware reliability, trustworthiness, safety and security are often hard to prove. A consequence of this hazard is the increasing importance and cost of software and hardware validation. However, traditional validation methods do not scale up easily, have increasing costs, are slow and so extend time to market and do not guarantee optimal quality.

Abstract interpretation is a general theory and practice for the semantic approximation of dynamic system's behaviour. A profound unity of all well known software validation methods such as static program analysis, (abstract) model checking, type and effect systems is clearly exhibited by their formalization in terms of abstract interpretation. They all aim at obtaining safe but approximate information about the behaviour of a computer system by systematically abstracting the semantics of its specification, model or program. Abstract interpretation not only formalizes these various abstractions but also provides guidelines for their refinement and combination, and it is also the base technology for the design of tools for verification of such properties at various levels of abstraction. In the last decade this technology has gone well beyond the standard academic circles, showing a great potential as a cutting-edge technology for industrial applications. Examples are: the AbsInt SME (www.absint.com) commercializing aiT an abstract interpretation-based tool that is able to determine precise upper bounds on the execution time of real-time programs running on processors used in embedded systems from aeronautics and automotive industries; the Polyspace (www.polyspace.com) component in MathWorks packages providing a semantics-based tool for the automatic analysis of embedded software in Ada, C and C++, for debugging run-time errors; the "Astrée" static analyzer (www.astree.ens.fr) which was able to verify automatically the absence of run-time errors in a safety-critical, real-time, synchronous, flight control program counting over 500.000 lines of C code embedded in civil European aircrafts (e.g., AIRBUS A380 being the latest). Another sign of the times is provided by the GlobalGCC (GGCC) project within the ITEA programme (Information Technology for European Advancement), whose goal is to extend the GNU Compiler Collection (GCC) with program-wide static analysis techniques. While the time is ripe, there are still significant problems to be solved before this revolution in (critical) software development can happen on a larger scale.

The goal of the AIDA2007 project is to strengthen the Italian research community in abstract interpretation, by promoting abstract interpretation as a basic enabling technology required to build dependable software systems. Our goal is to mobilize, unify, galvanize and strengthen the abstract interpretation community in Italy, with the main goal of creating a virtual laboratory with common scientific objectives, both in basic and applied research. This will be achieved by a better integration of the existing research capacities on abstract interpretation in Italy, notably in Padova, Parma and Verona, including the leading Italian research groups in abstract interpretation. Abstract interpretation will serve as the foundational theory to design methodologies and to build tools to ensure the interoperability, composability, scalability, reliability, robustness, dependability, security and autonomous self-adaptation of software.

As far as basic abstract interpretation research is concerned, we are interested in abstract domain and semantics design. It is well known that the the properties of an abstract interpretation strongly depend upon the properties of the underlying semantics and the properties of the chosen abstraction. We will approach this fundamental problem by studying abstractions and semantics from the viewpoint of the properties the resulting abstract interpretation has to meet. This is the case in the systematic design of optimal domains for model checking, the design of abstract domains for points-to and aliasing analysis of C and Java programs, the design of scalable and precise numerical abstract domains, the development of systematic methods for measuring the precision of an analysis by improving domains with probabilistic and statistical information, and semantic transformations for achieving completeness. The common aspect in all these tasks is that they all boil down to a refinement process of abstract domains. The well established theory of abstract domain refinement and manipulation will provide a common infrastructure in order to develop new abstract domains that will enrich the state-of-the-art in this major field.

As far as the domain of applications is concerned, we are interested in exploring new directions in static program analysis, code protection and system verification. In static program analysis, we are mostly interested in the new directions recently provided by complexity and termination analysis of C and Java code, and in causality analysis. We will study techniques to automatically prove the termination or bound the complexity of program fragments by exploiting suitable metric information in abstract domains. A similar domain-theoretic approach will be used to model causality in concurrent and distributed systems, in particular in CCS and pi-calculus, by modeling dependencies between process actions, both in terms of their spatial structure and their communication abilities. While static analysis is a well established domain where abstract interpretation plays a key role since decades, this is not the case in system verification and in the emerging field of code protection. In system verification we are again interested in abstraction refinement algorithms, in language transformations and in efficient algorithms for computing behavioural equivalences. Our goal here is to investigate how abstract interpretation can be applied to the context of CounterExample-Guided Abstraction Refinements (CEGAR) of abstract models, providing more precise algorithms for abstraction refinements in abstract model checking. Symmetrically, we will study language refinements and transformations avoiding spurious counterexamples. These research lines have the common goal of exploiting abstract interpretation in order to provide more precise verification algorithms for large systems. In code protection, we will consider both the risks connected with code hacking (malicious software attack) and theft (malicious host attack). Malicious host attacks include sensible information leakage, knowledge extraction by static and dynamic analysis, program decomposition for code reuse, integrity corruption. Malicious software attacks include code hacking by program deformation, namely inserting malware behaviour by vulnerability exploitation. On the side of malicious host attacks, software piracy and software tampering are the major types of attacks. Software watermarking and code obfuscation are promising defense techniques against, respectively, software piracy and malicious reverse engineering. We are interesed in designing new robust techniques for code obfuscation and software watermarking by abstract interpretation. While these techniques can be used against reverse engineering and copyright infringement, they can also provide the ground base for malicious software attacks. The threat of malware attacks is an unavoidable problem in today computer security. We are interested in going beyond known syntactic signature-based detection towards more semantics-based and property-based detection methodologies.

12 - Stato dell'arte

Testo italiano

L'interpretazione astratta è una ben nota teoria generale ed unificante per l'approssimazione della semantica formale di sistemi computazionali. Inventata da Cousot e Cousot [CC77, CC79] alla fine degli anni '70 come framewok unificante per progettare e validare analisi statiche di programmi, l'interpretazione astratta ha guadagnato sempre maggiore popolarità come metodologia generale per la descrizione e la formalizzazione di computazioni approssimate in aree molto diverse dell'informatica come l'analisi statica di programmi, la trasformazione di programmi, la verifica di sistemi e programmi via model checking, i sistemi di tipi, l'analisi di sistemi mobili e distribuiti, la sicurezza, il constraint solving, etc. Come notevole recente esempio di applicazione dell'interpretazione astratta, ASTRÉE (http://www.astree.ens.fr/) è un analizzatore statico di programmi strettamente basato sulla teoria dell'interpretazione astratta che è riuscito a verificare formalmente l'assenza di errori a run-time nel software di controllo volo primario del sistema fly-by-wire dell'Airbus A340 (132,000 linee di codice C piuttosto peculiare). Ci stiamo avvicinando al punto in cui verranno inseriti analizzatori statici basati su interpretazione astratta negli ambienti di sviluppo dei principali linguaggi imperativi.

Gli obiettivi del progetto AIDA2007 sono strutturati in quattro WorkPackage (WP) che si focalizzano su precisi aspetti della progettazione e delle applicazioni dell'interpretazione astratta.

WP1: Progettazione di Interpretazioni Astratte
WP2: Analisi Statica mediante Interpretazione Astratta
WP3: Protezione del Codice mediante Interpretazione Astratta
WP4: Verifica di Sistemi mediante Interpretazione Astratta

I membri di AIDA2007 sono ben noti nella comunità internazionale di interpretazione astratta e posseggono un notevole background ed esperienza di ricerca su progettazione e applicazioni dell'interpretazione astratta, in particolare sugli argomenti dei quattro WP di AIDA2007.

WP1: Progettazione di Interpretazioni Astratte

I primi lavori di Cousot e Cousot [CC77, CC79] hanno gettato le fondamenta per una teoria dei domini astratti e dei trasformatori di domini astratti. Dopo questi lavori fondazionali, sono stati definiti numerosi operatori aventi lo scopo di migliorare la precisione, ridurre la complessità e decomporre i domini astratti in modo sistematico [CFGPR97, FGR96, GR97, GR99]. La completezza gioca un ruolo chiave in interpretazione astratta, ed in particolare in molti problemi riguardanti i raffinamenti di domini astratti. Le astrazioni complete possono sempre essere ottenute costruttivamente come mimini raffinamenti completi e massime semplificazioni complete di un generico dominio astratto [GRS00]. I corrispondenti trasformatori semantici, ovvero trasformazioni della semantica osservazionale che la rendono completa rispetto ad una certa astrazione, sono sinora sconosciuti. Recentemente la completezza è stata usata per modellare la non-interferenza e la perdita di informazione: "What you lose in precision is what you leak in observation" [BGM07, GM05]. Questi risultati si basano sulla generalizzazione mediante interpretazione astratta della nozione di non-interferenza di Goguen e Meseguer [GM84] che ha portato alla nozione di non-interferenza astratta [GM04]. Il più potente attaccante innocuo e la massima informazione rilasciata da un programma sono stati caratterizzati come trasformatori di domini astratti. Uno sforzo sostanziale nell`ultima decennio è stato anche dedicato al problema di raffinare in modo sistematico i domini in abstract model checking e in predicate abstraction [BPR02, BCDR04]. I membri di AIDA2007 hanno studiato varie problematiche collegate al problema di ottenere strong preservation in abstract model checking attraverso raffinamenti di domini astratti [GQ01, RT04, RT05a, RT06, RT07a].

L'analisi dei puntatori e dell'aliasing per i linguaggi imperativi è un passo importante, a volte essenziale, per migliorare la precisione di numerose altre analisi. Nonostante la vasta letteratura sull'argomento, ad oggi è difficile ottenere risultati sufficientemente precisi utilizzando una quantità ragionevole di risorse di calcolo [Ber07, Hin01]. Le analisi dei puntatori attualmente in uso per la compilazione ottimizzata, essendo indipendenti dal contesto e dal controllo di flusso, non forniscono un livello di precisione sufficiente ai fini della verifica dei programmi. D'altra parte, le analisi dipendenti da contesto e/o dal flusso proposte in letteratura non catturano tutti i costrutti di linguaggi come il C [Ber07], oppure sembrano essere state abbandonate [Deu94, EGH94]. I risultati preliminari sull'analisi di proprietà numeriche per programmi C realistici hanno mostrato l'esistenza di problemi di precisione e scalabilità. Se l'uso indiscriminato del dominio dei poliedri convessi impedisce la scalabilità, d'altra parte il ricorso al dominio degli intervalli o anche degli ottagoni può a volte portare ad un numero inaccettabile di falsi allarmi (ad es., per l'analisi di "string cleanness"). Una soluzione a tali problemi è quindi di primaria importanza.

WP2: Analisi Statica mediante Interpretazione Astratta

I difetti del software contribuiscono in maniera significativa al suo costo totale di sviluppo [RTI02], in particolare se individuati dopo la fase di rilascio del prodotto. I difetti dalle conseguenze più costose sono quelli che introducono vulnerabilità di sistema. A tal proposito, l'analisi sistematica di migliaia di segnalazioni ha permesso al CERT, l'autorevole ente che opera nel campo della sicurezza informatica (http://www.cert.org/), di evidenziare come, alla base dei problemi di sicurezza più comuni, vi sia un numero limitato di cause. Il caso più frequente è quello di errori di programmazione quali overflow durante l'accesso ai buffer o nei calcoli sui tipi interi, errori nella formattazione di stringhe ed altri errori nella gestione della memoria. È quindi necessario studiare, da un lato, metodologie di progettazione del software che aiutino a prevenire tali problemi, dall'altro, strategie di validazione del software che, tenendone conto, siano di supporto ad una loro efficace individuazione e rimozione. Naturalmente, un prodotto software immune da errori come quelli precedentemente menzionati potrebbe comunque avere altri difetti che influiscono sulla sua funzionalità ad un più alto livello di astrazione. In ogni caso, l'assenza degli errori "di basso livello" è un prerequisito irrinunciabile, mancando il quale qualunque analisi svolta ai livelli superiori non fornirebbe alcuna reale garanzia. Per favorire le ricadute a livello pratico, AIDA2007 focalizzerà l'attenzione sui linguaggi imperativi più diffusi, come C (con possibili estensioni a C++) e Java.

La maggior parte delle vulnerabilità presenti nei programmi C ha origine da errori di manipolazione di stringhe; l'assenza di tali errori ("string cleanness") è un prerequisito per la scrittura di programmi che abbiano un comportamento prevedibile e robusto. I programmi C sono particolarmente soggetti a tali errori per via dell'assenza di un tipo di dato stringa predefinito. Sebbene vi siano alcune analisi statiche per il rilevamento dei buffer overflow, molto lavoro rimane da fare per renderle efficaci [DRS03, Ell04]. L'analisi di terminazione per programmi C è un argomento di ricerca di primaria importanza. Il linguaggio C è ampiamente usato per lo sviluppo di dispositivi embedded, spesso caratterizzati da elementi di criticità (ad es., nell'automazione industriale) e per i quali la terminazione di ogni componente funzionale è una proprietà essenziale. Un dispositivo che mantiene una data interazione con l'ambiente può essere modellato da un ciclo composto da alcune fasi (lettura sensori, calcolo della reazione appropriata, invio dei comandi agli attuatori), per le quali la mancata terminazione sarebbe causa di malfunzionamenti gravi quanto un fallimento di sistema. Un altro esempio è fornito dai driver dei dispositivi hardware, la cui mancata terminazione può causare lo stallo del sistema operativo. È anche per questa ragione che, al fine di semplificare la verifica automatica dei driver del proprio sistema operativo, Microsoft ha iniziato un progetto di ricerca sull'analisi di terminazione dei programmi C [CPR06]. Assicurata la terminazione, il passo logico successivo è costituito dall'analisi di complessità, che consente di garantire, ad esempio, il rispetto delle scadenze temporali da parte dei sistemi software. Terminazione e complessità sono state studiate per programmi imperativi, ristretti a valori primitivi [CPR06, GST06] o nel caso di programmi iterativi numerici attraverso l'espressione della semantica dei programmi in forma polinomiale [C05]. L'estensione di questi metodi a programmi che usano dati allocati sullo heap e aggiornamenti distruttivi complessi rappresenta una grande sfida. Sviluppi recenti forniscono un modello preciso dello heap, con pair-sharing [SS05] e cyclicity [RS06], che è ora implementato in Julia [Sp05] sfruttando PPL [BHZ06]. Analogamente, l'analisi di complessità di linguaggi Java-like determina le classi di complessità dei metodi basandosi sull'induzione [AAGPZ07].

Gli approcci quantitativi all'analisi dei programmi hanno lo scopo di sviluppare tecniche che forniscono risposte approssimate dotate di una stima numerica dell'approssimazione introdotta. L'interpretazione astratta probabilistica [DW01] ridefinisce l'interpretazione astratta classica in un contesto probabilistico, fornendo, oltre all'analisi del caso pessimo [M00], anche l'analisi del caso medio [DHP06]. In interpretazione astratta probabilistica i domini sono spazi lineari normalizzati, l'astrazione è definita attraverso operatori lineari e la concretizzazione attraverso la nozione di un'inversa lineare generalizzata. La vicinanza metrica fornisce una stima numerica quantitativa della precisione dell'astrazione.

AIDA2007 si interesserà anche allo sviluppo di tecniche di analisi statica basate su modelli causali di sistemi concorrenti e distribuiti. In particolare, nel contesto delle algebre di processi, i modelli non interleaving per CCS e il pi-calculus sono stati formalizzati come reti di Petri, sistemi di transizione etichettati generalizzati e strutture ad eventi [Win82, CVY07]. Inoltre, i lavori [BS98, DP99, Cas95] mostrano come sia possibile usare le equivalenze comportamentali per catturare le dipendenze tra le azioni dei processi, sia in termini della loro struttura spaziale che delle loro capacità di comunicazione.

WP3: Protezione del Codice mediante Interpretazione Astratta

L'offuscamento e la steganografia di software sono tecnologie emergenti per la protezione della proprietà intellettuale del codice [CTL98, CT02]. In base alla natura del processo di estrazione, le metodologie esistenti per la steganografia (watermarking, fingerprinting e birth-marking) sono classificate come statiche e dinamiche [NCT02]. La distinzione tra statico e dinamico è spesso troppo grossolana per modellare accuratamente gli attaccanti, che potrebbero combinare analisi statica e dinamica ed utilizzare altri metodi d'analisi come model checking e interpretazione astratta [CC04]. Un noto risultato negativo dimostra l'impossibilità dell'offuscamento [B01], assumendo che il programma originale e quello trasformato abbiano comportamenti identici e che ogni programma offuscato diventi incomprensibile rispetto ad ogni attaccante. Nella realtà questi vincoli sono spesso rilassati. Metodologie pratiche per rendere difficile il reverse engineering possono confondere la fase di disassembly [LD05] o quella di decompilazione [CT98, CTL98]. In [DPG04] si può trovare un primo tentativo di dare un fondamento semantico all'offuscamento del codice. L'idea è che un offuscamento nasconde tutte quelle proprietà che non sono preservate dalla trasformazione. Questo approccio è stato applicato all'inserimento dei predicati opachi [DPG05, DPG06] dove la capacità di un attaccante nel riconoscere i predicati opachi è ridotta ad un problema di completezza dell'astrazione che modella l'attaccante [DPG05, DPG06]. Tipicamente un programma è classificato come infetto da un malware se esso contiene la firma del malware, cioè la sequenza di istruzioni che lo caratterizzano [Szor05]. Gli algoritmi basati sulle firme sono in grado di identificare malware conosciuti ma non nuovi malware o l'offuscamento di malware metamorfici [N97], dove ogni offuscamento cambia la sintassi del malware in modo da cambiarne la firma [J02, SF01]. Allo scopo di resistere al metamorfismo sono stati recentemente proposti: un identificatore di malware che tiene conto della semantica [CJ2005], un identificatore di malware basato su model checking [K05] e un identificatore di malware con firme semantiche basato su interpretazione astratta [DCJD2007].

WP4: Verifica di Sistemi mediante Interpretazione Astratta

Uno degli obiettivi di AIDA2007 è l'applicazione dell'interpretazione astratta alla verifica automatica basata su model checking [CES86, CGP99]. In particolare, siamo interessati allo studio della relazione tra interpretazione astratta e abstract model checking. L'approccio dell'abstract model checking [CGL94, CGP99] fornisce una soluzione al problema fondamentale della scalabilità del model checking, il ben noto problema dell'esplosione esponenziale degli stati [CGP99]. La metodologia comune di qualunque tecnica di abstract model checking è di eseguire l'algoritmo di verifica su un modello astratto ridotto A che approssima alcune proprietà di interesse del sistema concreto M. Idealmente, il modello astratto ridotto A dovrebbe essere fortemente preservante: una formula di specifica f vale su M se e solo se f vale nel modello astratto A. D'altronde, molti modelli astratti non sono fortemente preservanti [CGP99]. Quindi, la presevazione forte viene ottenuta mediante operazioni di raffinamento di un modello astratto approssimato iniziale. La soluzione standard al problema della preservazione forte consiste in raffinamenti del modello astratto che sono guidati da una specifica formula. In questo contesto, il raffinamento di astrazioni guidato da controesempi (CEGAR, dall'acronimo inglese), proposto dal gruppo di ricerca di Clarke [CGJLV03], è divenuto la metodologia standard. Esempi ben noti di model checker basati sull'approccio CEGAR sono BLAST [HJMS03], MAGIC [CGJLV03] e SLAM [BR02]. La relazione tra interpretazione astratta e abstract model checking è già stata studiata sotto vari aspetti [CC99, CC00, CGL94, DGG97], ed in particolare i membri di AIDA2007 hanno studiato la relazione tra preservazione forte in abstract model checking e completezza in interpretazione astratta [GR06, RT04, RT05a, RT06, RT07a].

Nota: *Le citazioni bibliografiche in ogni sezione del presente Modello A si riferiscono alla bibliografia inserita nei Modelli B delle tre unità di ricerca.*

Testo inglese

Abstract interpretation is well known as a general and unifying theory for approximating the formal semantics of computational systems. Invented by Cousot and Cousot [CC77, CC79] in the late 1970s as a unifying framework for designing and validating static program analyses, abstract interpretation has increasingly gained popularity as a general methodology for describing and formalizing approximate computations in such diverse areas of computer science as static program analysis, program transformation, system and program verification by model checking, type systems, analysis of mobile and distributed systems, security, constraint solving, etc. As a remarkable recent example of application of abstract interpretation, ASTRÉE (http://www.astree.ens.fr/) is a static program analyzer firmly based on abstract interpretation theory that was able to prove the absence of any run-time errors in the primary flight control software of the Airbus A340 fly-by-wire system (132,000 lines of rather special C code). We are approaching the point where static analyzers based on abstract interpretation can be included in the development environments of mainstream imperative programs.

The objectives of the project AIDA2007 are structured as four WorkPackages (WPs) that have a focus on specific issues of designing and applying abstract interpretation.

WP1: Abstract Interpretation Design
WP2: Static Analysis by Abstract Interpretation
WP3: Code Protection by Abstract Interpretation
WP4: System Verification by Abstract Interpretation

The members of AIDA2007 are well known in the international research community on abstract interpretation and have a strong research background and experience on designing and applying abstract interpretation, in particular on the subjects of the four WPs of AIDA2007.

WP1: Abstract Interpretation Design

Early work by Cousot and Cousot [CC79] laid the foundation for a theory of abstract domains and abstract domain transformers. After this fundamental work, a number of operators have been designed to systematically improve precision, reduce complexity and modularize analyses by refining, simplifying and decomposing abstract domains [CFGPR97, FGR96, GR97, GR99]. Completeness plays a key role in abstract interpretation, and in particular in many problems concerning abstract domain refinements. Complete abstractions can be always constructively designed as least complete refinements and the greatest complete restrictions of any abstract domain [GRS00]. No corresponding transformers are known for semantics, i.e. transformations of observable semantics making them suitable for

completeness under a given fixed abstraction. Completeness has been recently considered also as a model for non-interference and information leakage: "What you lose in precision is what you leak in observation" [BGM07,GM05]. This relies upon the generalization of Goguen and Meseguer [GM84] non-interference by abstract interpretation provided by abstract non-interference [GM04]. The idea is to consider attackers as static program analyzers (i.e. abstract interpretations) whose task is to reveal properties of secret data by statically analyzing public resources. The most powerful attacker unable to disclose confidential properties and the maximal amount of information disclosed by a program under attack have been characterized as abstract domain transformers. A substantial effort in the last decade has also been devoted to the problem of systematically refining domains in abstract model checking and in predicate abstraction. The members of AIDA2007 studied a number of issues related to the problem of achieving strong preservation of abstract model checking by abstract domain refinements [GQ01, RT04, RT05a, RT06, RT07a].

Aliasing/pointer analysis in imperative languages is a useful, sometimes essential step to improve the precision of several other analyses. Despite the vast literature on the subject, it is currently very difficult to obtain reasonably precise results while using a reasonable amount of resources for the analysis [Ber07, Hin01]. Flow- and context-insensitive pointer analyses in use for optimized compilation, while being able to scale to millions of lines of code, do not provide enough precision for the purposes of program verification. On the other hand, flow- and/or context sensitive analyses proposed in the literature either are unable to capture all the constructs of languages such as C [Ber07], or seem to have been abandoned for some (unknown) reason [Deu94,EGH94]. Preliminary results on the analysis of numerical properties of real C programs clearly showed that there are problems in both scalability and precision of the analysis. A blind use of the domain of convex polyhedra implies non scalability, while resorting to intervals or even octagons sometimes does not allow to acceptably reduce the number of false alarms (e.g., for string cleanness analysis). Solving these problems of scalability and precision is thus a priority.

WP2: Static Analysis by Abstract Interpretation

It is widely acknowledged that software defects significantly contribute to the high costs of software development [RTI02]. Moreover, defects that are detected only after the product has been released have an higher cost, both for producers and consumers. Software defects whose consequences are particularly expensive are those that introduce system vulnerabilities. After the thorough analysis of thousands of reports about computer systems vulnerabilities, CERT, the authoritative center for computer security (http://www.cert.org/), concluded that most of the actually exploited security problems are due to a limited number of causes. The most frequent ones originate from programming mistakes: out of bounds accesses to buffers, overflows in operations on native integers, string manipulation errors and other errors related to memory management. It is therefore important to study, on the one hand, software design methodologies that can help preventing these problems and, on the other hand, software validation strategies that, being aware of these problems, do help in their detection and removal. In most cases, of course, a software product unaffected by the aforementioned problems can still have defects that impact the user, as other and more abstract properties must be considered to uncover higher-level flaws. However, the absence of "low level" defects is clearly a prerequisite to the reliability of any higher-level analysis. In order to have an impact on this state of affairs, AIDA2007 focuses its research on mainstream imperative languages such as C (with possible extensions to C++) and Java.

Most of the vulnerabilities occurring in C programs stem from string manipulation errors. The absence of such errors, i.e., the string cleanness, is a prerequisite for secure programs that have a predictable behavior and are robust against malicious attacks. Programs in C are particularly prone to such errors due to the lack of a native string data type. Static analysis techniques for the detection of buffer overflows have been proposed, but work remains to be done in order to make them effective [DRS03, Ell04]. Termination analysis for C programs is a main topic in current static analysis research. The C language is widely used to develop embedded devices, often including critical aspects (in industrial control, automotive industry, etc.), where the proper termination behavior of each functional component is essential. Imagine a device whose goal is to maintain a certain type of interaction with the environment. This can be conceived as an infinite loop consisting of several phases (read sensors, compute a proper reaction, send commands to actuators), whose termination failure would cause malfunctions as bad as classical system failures. Another example comes from device drivers, where a termination failure can cause the whole operating system to hang. For this reason Microsoft started a research project to study automatic termination analysis of C programs: drivers for the Microsoft operating systems are developed by hundreds of hardware producers and their automatic verification is crucial to the US software giant [CPR06]. Once termination is guaranteed, complexity analysis is the logical step forward to ensure, e.g., the hard deadlines are met by software systems. Termination and complexity were studied for imperative programs, restricted to primitive values [CPR06, GST06] or in the case of numerical iterative programs by expressing program semantics in polynomial form [C05]. A great challenge is to extend these methods to programs using dynamically heap-allocated data and complex destructive updates. Recent developments provide a precise model of the heap, with pair-sharing [SS05] and cyclicity [RS06], which are now implemented in Julia [Sp05] by exploiting the PPL [BHZ06]. Similarly, complexity analysis of Java-like languages determines the complexity classes of the methods by inductive reasoning [AAGPZ07].

Quantitative approaches to program analysis aim at developing techniques which provide approximate answers together with numerical estimates of the approximation introduced by the analysis. Probabilistic abstract interpretation [DW01] recasts classical abstract interpretation in a probabilistic setting, providing, in addition to the standard worst-case analysis [M00], also an average-case analysis [DHP06]. In probabilistic abstract interpretation, domains are normed linear spaces, abstractions are defined via linear operators and concretisation via the notion of a linear generalised inverse. Metric closeness provides a quantitative numerical estimate of the precision of the abstraction.

AIDA2007 is also interested in the development of static analysis techniques based on causal models of concurrent and distributed systems. In particular, in the context of process algebras, non-interleaving models for CCS and the pi-calculus have been formalized as Petri nets, generalized labelled transition systems, and event structures [Win82, CVY07]. Moreover, [BS98, DP99, Cas95] show how behavioral equivalences can be used to capture the dependencies between process actions, both in terms of their spatial structure and their communication abilities.

WP3: Code Protection by Abstract Interpretation

Code obfuscation and software steganography are emerging technologies for code protection in modern software design and engineering [CTL98, CT02]. Most of the existing code steganography techniques -- watermarking, fingerprinting and birth-marking -- are classified either as static or dynamic, according to the nature of their extraction processes [NCT02]. The distinction between the static and dynamic nature of code steganography is often too rough to provide an accurate model of attackers, which may combine static and dynamic code analysis techniques as well as other analysis methodologies such as abstract interpretation and model checking [CC04]. An impossibility result is known for code obfuscation [B01] under the assumption that the original and transformed program have identical behaviours and where any transformed program becomes unintelligible to any adversary. In practical contexts these constraints are often relaxed. Practical approaches to make reverse engineering hard can either confuse the disassembly process [LD05] or the decompilation process [CT98, CTL98]. A first attempt towards a semantics-based foundation of code obfuscation is in [DPG04]. The idea is that an obfuscator hides all those properties that are not preserved by the transformation. This approach as been applied to opaque predicate insertion [DPG05, DPG06] where the ability of an attacker in disclosing opaque predicates is reduced to a completeness problem of the abstraction that models the attacker [DPG05,DPG06]. Misuse detection classifies a program as infected by a malware when the malware signature, i.e. the sequence of instructions characterizing the malware, occurs syntactically in the program [Szor05]. In general, signature-based algorithms detect known malware, but they are ineffective against unknown malicious programs or obfuscated code in metamorphic malware [N97]. The basic idea of metamorphism is that each successive transformation of a malware changes the syntax in order to foil misuse detection [J02, SF01]. Recently, a semantics-aware malware detector [CJ2005], a malware detector based on standard model checking [K05], and an abstract interpretation-based malware detector for semantic signatures [DCJD2007] have been proposed in order to make malware detection insensitive to metamorphism.

WP4: System Verification by Abstract Interpretation

One objective of AIDA2007 is to apply abstract interpretation to automatic system verification based on model checking [CES86, CGP99]. In particular, we are interested in studying the relationship between abstract interpretation and abstract model checking. The abstract model checking approach [CGL94, CGP99] provides a solution to the critical issue of model checking scalability, the well-known state explosion problem [CGP99]. The common methodology of any approach to abstract model checking is to run the verification algorithm on a reduced abstract model A that approximates some properties of interest of the concrete system model M. Ideally, the reduced abstract model A should be strongly preserving: a specification formula f holds on M iff f is satisfied on the abstract model A. However, most abstract models are not strongly preserving [CGP99]. Thus, strong preservation is achieved by performing some refinement action of an initial rough abstract model. The standard solution to the strong preservation problem consists in abstract model refinements that are driven by a given specification formula. In this context, CounterExample-Guided Abstraction Refinement (CEGAR), as pionereed by Clarke et al. [CGJLV03], has become the standard methodology. Well-known examples of model checkers that follows the CEGAR approach are BLAST [HJMS03], MAGIC [CGJLV03] and SLAM [BR02]. The relationship between abstract interpretation and abstract model checking has already been investigated [CC99, CC00, CGL94, DGG97], in particular the members of AIDA2007 investigated the relationship between strong preservation in abstract model checking and completeness in abstract interpretation [GR06, RT04, RT05a, RT06, RT07a].

Remark: *The bibliographic citations in each section of this document "Modello A" refer to the bibliography included in the three documents "Modello B" for Padova,*

Parma and Verona units.

13 - Articolazione del Progetto e tempi di realizzazione

Testo italiano
Il piano dei lavori della presente proposta di progetto AIDA2007 è organizzata nei seguenti quattro "WorkPackage" (WP):

WP1: Progettazione di Interpretazioni Astratte
WP2: Analisi Statica mediante Interpretazione Astratta
WP3: Protezione del Codice mediante Interpretazione Astratta
WP4: Verifica di Sistemi mediante Interpretazione Astratta

Ogni WP si focalizza su aspetti generali o specifici della progettazione e delle applicazioni dell'interpretazione astratta. Ci proponiano di avanzare lo stato dell'arte della ricerca sulle tematiche comprese da ogni WP portando a termine un congruo numero di precisi "Task" di ricerca. Ogni Task è assegnato ad una delle tre unità di ricerca: Padova (PD), Parma (PR) e Verona (VR). Ad esempio, il Task PD-1.2 denota il Task di ricerca dell'unità di Padova individuato dalla numerazione 2 all'interno del WP1. Nel seguito diamo una sommaria descrizione delle attività di ricerca per ogni Task in cui è articolato il progetto.

WP1: Progettazione di Interpretazioni Astratte

- Task PD-1.1: Domini astratti per il model checking
Mentre l'approccio standard all'abstract model checking prevede l'uso di un sistema di transizione astratta, l'interpretazione astratta permette di definire un abstract model checker che si basa su modelli astratti definiti come generici domini astratti ed in particolare su domini astratti disgiuntivi. Ci prefiggiamo di investigare se e come la classe dei domini astratti disgiuntivi possa essere utile in altri contesti applicativi, in particolare per gli algoritmi di abstract refinement. Intendiamo inoltre studiare nuove classi di domini astratti che possano essere usate con profitto in abstract model checking.

- Task PR-1.1: Domini astratti per l'analisi dei puntatori e dell'aliasing in programmi C e Java
La maggior parte della ricerca sulle analisi dei puntatori e dell'aliasing è focalizzata sulla compilazione ottimizzata e, in tale contesto, solo le analisi più efficienti sono considerate utili. Alcune sperimentazioni hanno però mostrato che la precisione di tali analisi è insufficiente per la verifica automatica dei programmi. Studieremo la possibilità di regolare tale compromesso tra precisione ed efficienza al fine di consentirne l'applicabilità nel contesto della verifica.

- Task PR-1.2: Domini numerici scalabili e precisi
Contrariamente a quanto detto per PR-1.1, per le proprietà numeriche esiste una vasta scelta di domini astratti con vari livelli di precisione ed efficienza ed implementazioni di buona qualità. Come per PR-1.1, i problemi di scalabilità impediscono l'adozione diretta dei domini più precisi, mentre considerazioni sulla precisione suggeriscono di evitare i domini più efficienti, se non quando le altre opzioni sono impraticabili. Studiremo combinazioni scalabili dei domini esistenti, cercando un mix adeguato tra operatori approssimati, widening e tecniche di partizionamento delle variabili

- Task VR-1.1: Trasformazioni semantiche orientate alla completezza
Siamo interessati alla progettazione di trasformazioni che rendano complete le semantiche rispetto ad una data astrazione. La completezza è raggiunta calcolando la semantica più simile alla semantica di partenza che sia al contempo completa per l'astrazione data. Crediamo che il problema si risolva sotto condizioni non restrittive per completezza sia backward che forward [GQ01] e che l'esistenza e l'unicità della semantica da noi ricercata sia garantita in caso tanto di sovra- quanto di sotto-approssimazioni.

- Task VR-1.2: Misura di precisione basata su domini numerici
Spesso, per analizzare i possibili valori (numerici) delle variabili di un programma, si usano domini astratti -- ad esempio il dominio degli intervalli -- che danno risultati imprecisi. L'imprecisione degli intervalli è dovuta alla perdita delle relazioni tra variabili e alla rappresentazione dei valori assunti da una variabile tramite i soli minimo e massimo, che non permettono ad esempio l'individuazione di segmenti "proibiti" all'interno degli intervalli. Una delle analisi più precise si basa sul dominio dei poliedri [CH78], in cui un insieme di valori è rappresentato da un insieme di vincoli lineari. Riducendo l'espressività dei vincoli, si ottengono domini relazionali più deboli, come il dominio degli ottagoni. Chiaramente, una volta quantificata la precisione di un dominio, si può scegliere il dominio più appropriato per l'analisi di un'applicazione in base ad un trade-off con fattori di costo (come il tempo d'esecuzione), e non solo alla correttezza dell'analisi. Nell'analisi statica classica, l'imprecisione di un'analisi rispetto ad un programma è di solito valutata mediante l'estremo inferiore e superiore dell'insieme dei falsi positivi. Questa stima è tuttavia piuttosto approssimativa. Intendiamo quindi usare l'interpretazione astratta probabilistica al fine di introdurre una maggiore sistematicità nella scelta dei domini astratti.

WP2: Analisi Statica mediante Interpretazione Astratta

- Task PD-2.1: Semantiche causali di sistemi concorrenti e distribuiti
Le strutture ad eventi sono diffusamente usate per modellare la "true-concurrency" nei calcoli di processi. Intendiamo specificare una semantica composizionale per il "free name passing" nel pi-calculus che sia basata su un modello a strutture di eventi. Lo scopo è quello di ottenere una semantica denotazionale "fully abstract" per il full pi-calculus. Siamo inoltre interessati allo studio di semantiche composizionali simili per linguaggi imperativi multithreaded che, se comparate alle algebre di processi, si focalizzeranno sullo stato piuttosto che sul controllo di un thread.

- Task PD-2.2: Analisi statica di proprietà causali
Sfruttando i modelli "true-concurrent" di sistemi concorrenti e distribuiti basati su strutture ad eventi, siamo interessati a sviluppare tecniche di analisi statica per proprietà causali di processi concorrenti. In particolare, intendiamo studiare la causalità e la distribuzione spaziale dei processi che possono essere osservate tramite un modello di struttura ad eventi. Ci prefiggiamo infine di studiare il problema del model checking su strutture ad eventi.

- Task PR-2.1: Definizione di uno schema di analisi modulare e generico
I framework di analisi esistenti per i programmi imperativi impongono eccessive restrizioni al linguaggio analizzato, oppure mancano di una vera e propria modularità, impedendo di sfruttare le caratteristiche avanzate dei domini astratti più sofisticati. Basandosi su lavori esistenti, progetteremo un framework immune da tali inconvenienti.

- Task PR-2.2: Annotazione di programmi
I linguaggi C, C++ e Java non consentono al programmatore o agli strumenti di supporto alla programmazione di annotare adeguatamente i programmi con informazioni (dedotte in modo automatico o fornite dal programmatore) utili alla loro effettiva comprensione. Estensioni del linguaggio che permettano annotazioni dalla semantica chiara sono di sicuro interesse e saranno quindi oggetto di studio.

- Task PR-2.3: Analisi di correttezza della manipolazione di stringhe
Gli errori nella manipolazione di stringhe C sono la causa di buona parte dei bachi che affliggono il software odierno e causano molte vulnerabilità di sistema. Studieremo approcci modulari nei quali il problema di dimostrare l'assenza di errori di manipolazione di stringhe viene tradotto in un problema per la cui soluzione sono sufficienti analisi che tracciano il valore di variabili intere, booleane e puntatore.

- Task PR-2.4: Analisi di terminazione e complessità
Le analisi di terminazione e complessità di programmi C sono importanti argomenti di ricerca: ad esempio, svolgono un ruolo cruciale nella verifica delle componenti dei sistemi reattivi e dei device driver. Studieremo tecniche, anch'esse basate sull'analisi dei puntatori e delle variabili numeriche, per ottenere automaticamente dimostrazioni di terminazione o limitazioni della complessità di frammenti di programmi.

- Task VR-2.1: Analisi statica di terminazione e complessità

L'attuale piattaforma analisi di terminazione basata su Julia e Binterm e` gia` in grado di dimostrare la terminazione di piccoli programmi Java in bytecode, che possono anche includere strutture dati. Intendiamo estendere il nosto analizzatore in modo che possa trattare programmi realistici di medio/grandi dimensioni. L'analisi di complessita` di programmi Java in bytecode si basera` su delle analisi preliminari e di path-length gia` implementate nel tool Julia e usera` tecniche simili a quelle introdotte in [AAGPZ07], che ancora necessitano di una implementazione e di una valutazione sperimentale.

- Task VR-2.2: Ottimizzazione speculativa
L'idea è di (1) eseguire il codice in modo "speculativo" rispetto a un certo "guess"; (2) verificare successivamente la correttezza del "guess"; (3) se è corretto la computazione continua, altrimenti viene eseguito del codice riparatore. Questo garantisce che la computazione sia sempre eseguita correttamente. Il problema principale è il costo (medio) della "riparazione". Se il "guess" è spesso corretto allora il costo di esecuzione del codice riparatore è ammortizzato. Quindi, nell'approccio speculativo il compilatore può sfruttare anche dell'informazione statistica (piuttosto che solamente definita) per scegliere se fare l'ottimizzazione. L'interpretazione astratta probabilistica permette di sfruttare meglio il caso del "forse" rispetto alla classica analisi conservativa di ottimizzazione dei compilatori, sostituendo un'analisi di possibilità con una di probabilità.

WP3: Protezione del Codice mediante Interpretazione Astratta

- Task VR-3.1: Generalizzazione di firme
Il problema principale degli algoritmi di identificazione di malware basati su firme è che non sono in grado di riconoscere malware offuscati. Per affrontare questo problema forniremo un metodo di generalizzazione della firma in modo da poter riconoscere anche le varianti metamorfiche. L'intento maligno di un malware può essere espresso tramite particolari sequenze di system call che possono essere modellate come un linguaggio riconosciuto da un automa.

- Task VR-3.2: Predicate obfuscation
Vogliamo studiare quali siano i reali limiti implicati dal noto risultato di Barak et al. [B01] sull'impossibilità dell'offuscamento. Tale risultato si basa sul fatto che l'offuscamento debba nascondere ogni proprietà di programmi che non è derivabile dall'osservazione del loro comportamento input-output. Questa è chiaramente una richiesta molto forte mentre, in pratica, si è interessati a proteggere proprietà particolari di particolari programmi. Sarebbe quindi interessante studiare la possibilità di nascondere una specifica proprietà di un particolare programma (o classe di programmi).

- Task VR-3.3: Inserimento di software watermark in cicli
Un ciclo è un costrutto di programmazione eseguito iterativamente. Dispiegando le sue iterazioni come codice sequenziale, rendiamo il comportamento del ciclo ad ogni iterazione sintatticamente analizzabile. Così la trasformazione inversa diviene un modo per ridurre la comprensione del comportamento di ciascuna iterazione. L'idea è di sfruttare questo fatto per l'inserimento, all'interno di un ciclo, di watermark calcolati iterativamente. Questo Task richiederà tecniche di program slicing generalizzate (e astratte) in grado di gestire anche dipendenze approssimate. Su tali tecniche infatti si basa, almeno parzialmente, l'estrazione del watermark.

- Task VR-3.4: Nascondere watermark in lacune di completezza
*Le astrazioni complete modellano la precisa comprensione della semantica di programmi da parte di un osservatore approssimato, e corrispondono alla possibilità di sostituire, senza perdita di precisione, le computazioni concrete con quelle astratte. La mancanza di completezza dell'osservatore corrisponde quindi ad una imprecisa comprensione della semantica. Consideriamo lo statement C: x = a*b. Un osservatore (passivo) analizzando il segno delle variabili del programma e conoscendo il segno di a e b ha una comprensione completa del segno di x. Un'analisi statica dei segni che sostituisce il calcolo concreto con quello approssimato (i.e. la regola dei segni) è quindi in grado di conoscere, senza perdita di precisione, il valore del segno di x poichè l'astrazione dei segni è completa per la moltiplicazione intera. Una trasformazione t che sostituisce C con t(C): x=b; while b!=0 {x=a+x; b=b-1;} offusca l'informazione di segno all'osservatore. Questo perché la regola dei segni è incompleta per l'addizione intera. L'idea e` di sfruttare le lacune di completezza per nascondere/proteggere dati sensibili, i.e. software watermarking/offuscamento di codice.*

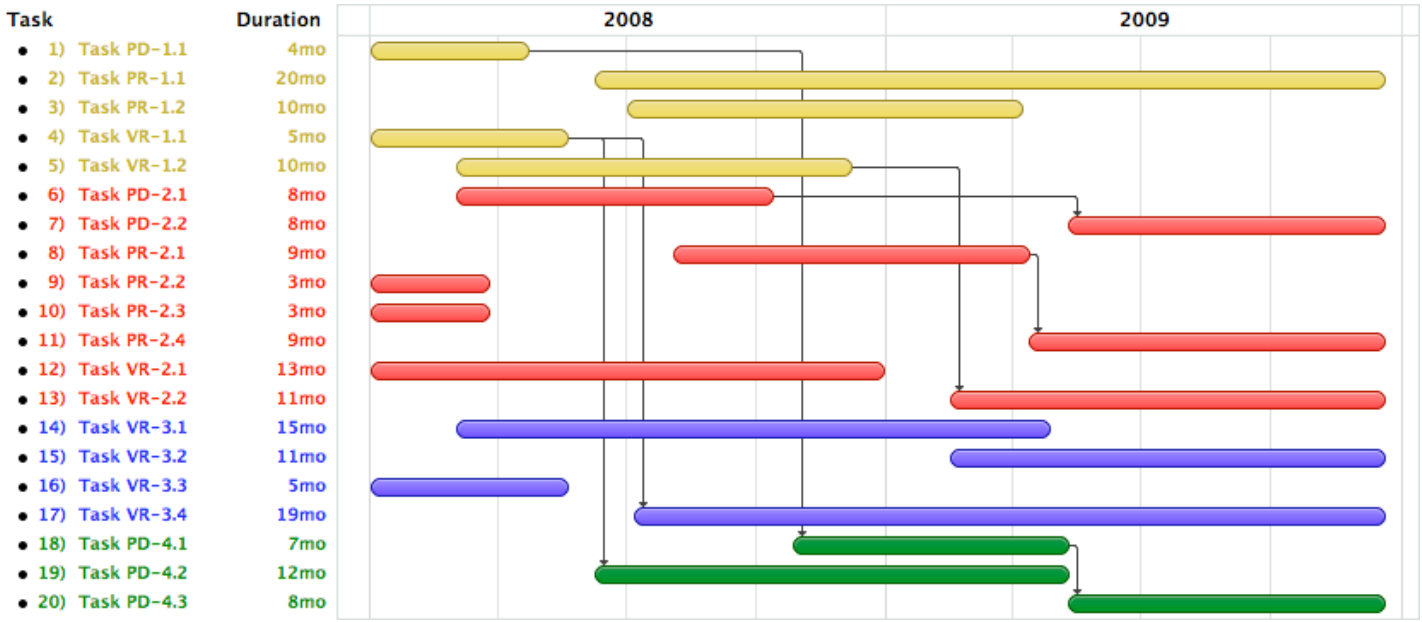
WP4: Verifica di Sistemi mediante Interpretazione Astratta

- Task PD-4.1: Algoritmi di raffinamento di astrazioni
Il raffinamento di modelli astratti guidato da controesempi (CEGAR, dall'acronimo inglese) è una tecnica molto diffusa e di successo per la progettazione di model checker. Intendiamo studiare come l'approccio generico basato su interpretazione astratta possa essere applicato nel contesto di model checker basati su CEGAR. In particolare, miriamo a generalizzare il metodo CEGAR a modelli astratti più generali specificati mediante interpretazione astratta ed a generici linguaggi di specifica. Vogliamo inoltre studiare il ruolo dei domini astratti completi negli algoritmi di abstraction refinement.

- Task PD-4.2: Trasformazioni di linguaggi
La proprietà di strong preservation viene ora raggiunta mediante un qualche raffinamento del modello astratto. Il nostro obiettivo è di modificare il linguaggio di specifica piuttosto del modello astratto per ottenere strong preservation. Intendiamo affrontare questo problema usando tecniche tipiche dell'interpretazione astratta, in particolare raffinamenti e semplificazioni di domini astratti. Come applicazione particolarmente rilevante, ci proponiamo di studiare come un tale metodo possa essere usato nel contesto del problema "branching vs linear" nel potere espressivo dei linguaggi temporali.

- Task PD-4.3: Algoritmi di calcolo di equivalenze comportamentali
La maggior parte dei raffinamenti di modelli astratti per ottenere strong preservation rispetto ad un intero linguaggio sono riconducibili al calcolo di qualche equivalenza comportamentale. In particolare, l'equivalenza di simulazione corrisponde alla strong preservation del linguaggio ACTL. Abbiamo recentemente individuato un nuovo algoritmo di calcolo dell'equivalenza di simulazione che migliora la complessità di tempo sinora conosciuta per questo problema. Ci proponiamo di investigare se le tecniche usate per definire questo nuovo algoritmo per l'equivalenza di simulazione possano essere generalizzate ed adattate per altre equivalenze comportamentali come la bisimulazione branching.*

Il seguente diagramma di Gantt mostra una stima delle date di inizio e delle durate dei venti Task previsti da AIDA2007, assieme alle relazioni di dipendenza tra Task diversi, assumendo che tale progetto biennale venga realizzato durante gli anni 2008 e 2009.



Testo inglese
The workplan of the present project proposal AIDA2007 is organized in the following four WorkPackages (WPs):

WP1: Abstract Interpretation Design
WP2: Static Analysis by Abstract Interpretation
WP3: Code Protection by Abstract Interpretation
WP4: System Verification by Abstract Interpretation

Each WP concentrates on generic or specific issues of designing and applying abstract interpretation. We plan to advance the state-of-the-art of the research that falls within each WP by accomplishing a number of focussed research tasks. Each task is assigned to a research unit: Padova (PD), Parma (PR) and Verona (VR). For example, Task PD-1.2 stands for the research task of the Padova unit numbered 2 within WP1. The following description provides an overview of the planned research activites within each WP.

WP1: Abstract Interpretation Design

- Task PD-1.1: Abstract domains for model checking
While standard abstract model checking frameworks rely on abstract models given as abstract transition systems, i.e. an abstract state space A and an abstract transition relation over A, abstract interpretation allows to design abstract model checkers that rely on generic abstract domains and in particular on disjunctive abstract domains. We plan to investigate whether and how disjunctive abstract domains can be useful in other contexts, notably in abstraction refinement algorithms. Moreover, we plan to study new classes of abstract domains to be used in abstract model checking.

- Task PR-1.1: Abstract domains for points-to and aliasing analysis of C and Java programs
Most of the research on points-to and aliasing analysis focuses on optimized compilation and, for that application, only very quick analyses have been considered worthwhile. However, preliminary investigations show that the precision attainable by that analyses is insufficient for the purposes of program verification. We will investigate whether the complexity/precision trade-off can be solved in ways that are adequate for the latter applications.

- Task PR-1.2: Scalable and precise numeric abstract domains
Differently from the case of PR-1.1, there is a wide choice of well-studied abstract domains with various degrees of precision and efficiency and high-quality implementations. Similarly to the case of PR-1.1 hard scalability problems (such as an exponential complexity) prevent the blind adoption of the most precise domains, whereas precision considerations suggest to adopt the most efficient domains only in restricted contexts, when all else fails. We will study scalable combinations of the existing domains by integrating approximate operators, widenings and partitionings of the variables' space.

- Task VR-1.1: Completeness-driven semantic transformations
We are interested in designing transformational methods for making semantics complete with respect to a given fixed abstraction. Completeness here means that the semantics is transformed towards the closest one which is complete for the given abstraction. We believe that this problem can be solved under nonrestrictive conditions for all forms of completeness (both backward and forward [GQ01]) and that the existence of the unique closest semantics which is complete for a given abstraction is guaranteed for both upper and lower approximations.

- Task VR-1.2: Measuring precision in numerical domains
When one tries to analyse the possible (numerical) values of variables in a program, certain commonly used abstract domains -- like for example the lattice of intervals -- can give rather imprecise results. In the case of intervals this is due to both the abstraction from the relationship between variables and the representation of a set of (possible) values by only its smallest and largest element which makes it impossible to discover, for example, "forbidden sections" within an interval. One of the most precise analyses is based on the polyhedron domain [CH78], where a set of values is represented by a set of linear constraints. By reducing the expressiveness of the constraints one can obtain weakly relational domains such as the octagon domain. Clearly, if we are able to quantify the precision of a domain we can select the domain which is more appropriate for a specific application on the basis of the possible trade-off with other costs (run-time, or other resource consumption), rather than simply of the correctness of the analysis result. In classical static analysis a standard approach to evaluating the imprecision of a static analysis is via the lower and upper bound of the set of false positive. This is however a quite rough estimate of the actual percentage of false positives for one program and analysis. We will exploit probabilistic abstract interpretation for introducing a more systematic approach in the selection of abstract domains.

WP2: Static Analysis by Abstract Interpretation

- Task PD-2.1: Causal semantics for concurrent and distributed systems
Event structures are widely used for modelling "true-concurrency" in process calculi. We plan to specify a compositional semantics for free name passing in the pi-calculus that is based on event structures. We aim at obtaining a fully abstract denotational semantics for the full pi-calculus. Moreover, we are also interested in studying similar compositional semantics for multithreaded imperative languages.

- Task PD-2.2: Static analysis of causal properties
Relying on true-concurrent models of concurrent and distributed systems based on event structures, we are interested in developing techniques for the static analysis

of causal properties of concurrent processes. In particular, we plan to study the causality and the spatial distribution of processes that can be observed through an event structure model. Moreover, we plan to study the model checking problem over event structures.

- Task PR-2.1: Definition of a modular and generic analysis framework
Existing analysis frameworks for imperative programs either put too strong restrictions on the analyzed languages or are not truly modular, preventing the exploitation of the characteristics of the more sophisticate analysis domains. Building on existing work, we will design a framework that tries to overcome all these difficulties.

- Task PR-2.2: Annotation of programs
The languages that are the subject of our research, mainly C, C++ and Java, do not allow programmers and programming tools to annotate programs with definite knowledge (whether it be automatically inferred or provided by the programmer), intentions and expectations. Language extensions that allow this kind of assertions with a well-defined semantics are interesting for a number of reason: we will work in this direction.

- Task PR-2.3: Analysis of string cleanness
Errors in the manipulation of C strings are the source of a significant proportion of the bugs that plague today's software and are responsible of many vulnerabilities. We will study modular ways whereby the problem of ensuring that a program is clean from string manipulation errors it translated into a problem whose solution can be approximated by analyses tracking the values of integer, Boolean and pointer variables.

- Task PR-2.4: Analysis of termination and complexity
Termination and complexity analyses of C programs are important topics: termination of the individual components of reactive systems and termination of device drivers are two examples where it plays a crucial role. We will study techniques, again based on the analysis of pointers and numerical variables, to automatically prove the termination or bound the complexity of program fragments.

- Task VR-2.1: Static analysis for termination ad complexity
Our current termination analysis based on Julia and Binterm is already able to prove termination of small Java bytecode programs, also dealing with data structures. We plan to extend it to the analysis of large, real programs. Automatic complexity analysis of Java (bytecode) programs will be based on the same preliminary and path-length analyses already implemented in Julia and will use techniques similar to those in [AAGPZ07], which still lack an implementation and an evaluation.

- Task VR-2.2: Speculative optimisation
The idea here is (1) to execute code "speculatively" based on some "guess"; (2) then to test at a later stage whether the guess was correct; (3) if it was correct the computation continues, otherwise some repair code is executed. This guarantees that the computation is always correctly performed. The only issue concerns the (average) costs of "repairs". If the guess is correct sufficiently often, the execution of the repair code for the cases where it is wrong is amortised. Thus, in the speculative approach the compiler can take advantage of statistical (rather than only definite) information when choosing whether to perform an optimisation. By using probabilistica abstract interpretation we can achieve a better exploitation of the "maybe" case than with classical conservative compiler optimisation, replacing a possibilistic analysis by a probabilistic one.

WP3: Code Protection by Abstract Interpretation

- Task VR-3.1: Signature generalization
The major problem of misuse malware detection techniques is that they are not able to recognize obfuscated variants of malware. A possible way to address this problem is to propose a methodology for generalizing a given malware signature in order to recognize also its metamorphic variants. The idea is to start with an initial description of a malware and then to generalize it in order to detect also its obfuscated variants. The malicious intent of a malware can be expressed in terms of particular sequences of system calls that can be modeled as the language recognized by an automaton.

- Task VR-3.2: Predicate obfuscation
We will investigate the true limits implied by the well-known negative result of Barak et al. [B01] on the impossibility of code obfuscation. Following this definition, obfuscation should hide every property of the program that is not derivable from its input-output behaviour. It is clear that this is a very strong definition while, in practice, we are usually interested in protecting only particular predicates (sensible properties) of particular programs. Hence, it would be interesting to investigate the possibility of hiding a precise predicate of a particular program (or class of programs).

- Task VR-3.3: Hiding software watermarks in loop structures
A loop is a programming construct whose execution is arranged in iterations. Unfolding a loop means deploying its iterations as sequential equivalent code, so that loop behavior at each iteration becomes syntactically analyzable. Thus the inverse of unfolding, namely folding, is a way to dim the comprehension of what each iteration exactly does. We can exploit this fact to inlay inside loops iteratively constructed watermarks. This task will require the generalization of standard program slicing techniques to cope with abstraction and approximate dependencies. Such generalized (abstract) program slicing will be used for extracting watermarks.

- Task VR-3.4: Hiding watermarks in completeness holes
Complete abstractions model precisely the complete understanding of program semantics by an approximate observer, which corresponds to the possibility of replacing, with no loss of precision, concrete computations with abstract ones. The lack of completeness of the observer is therefore the corresponding of its poor understanding of program semantics, and provides the key aspect for designing a new family of methods and tools for well-defined and possibly provable secure practical steganography and more in general obfuscation. In this context, obfuscation and incompleteness are deeply related concepts: While steganography can be seen as hiding information in completeness holes, code obfuscation can be analogously seen as transforming programs in such a way that possible observers cannot use complete abstractions for code analysis unless cost expensive analysis, e.g., complex relational or exponential analyses on the number of variables, are used. We are interested in studying both passive and active attacks performed by observers and use the results of Task VR-1.1 in order to provide a model for this behaviour.

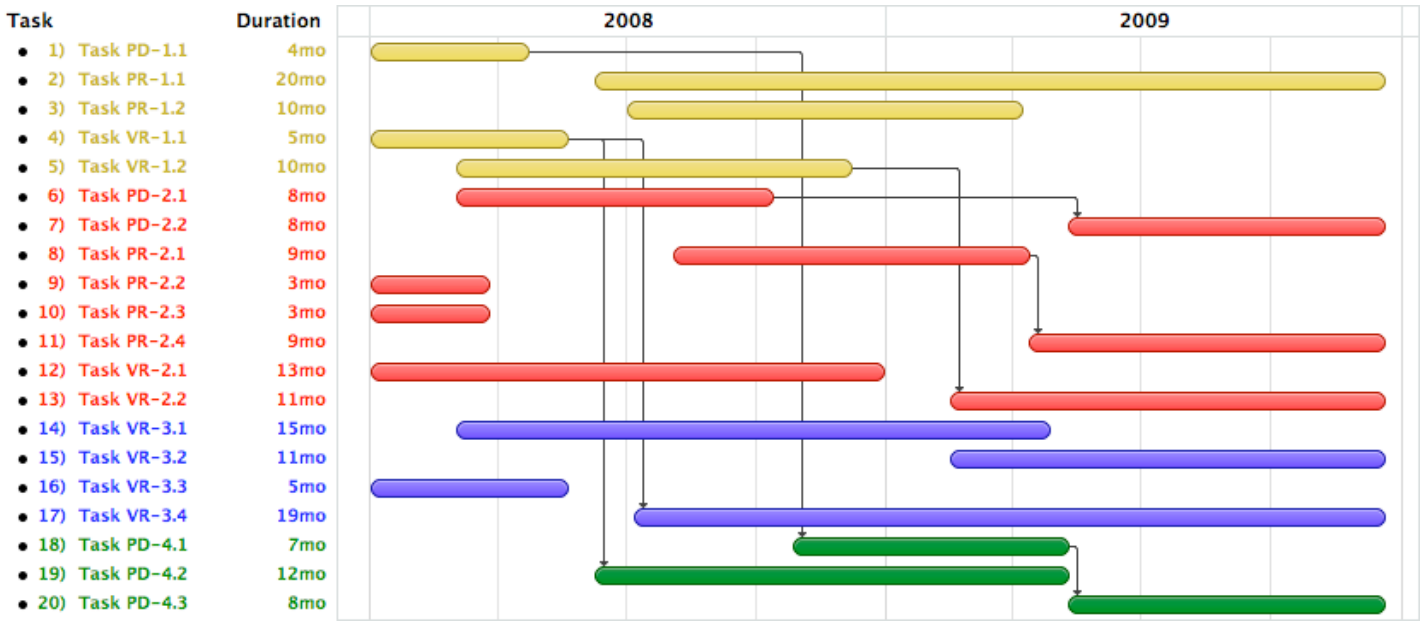
WP4: System Verification by Abstract Interpretation

- Task PD-4.1: Abstraction Refinement Algorithms
Counterexample-guided abstraction refinement has become a very popular and successful approach for designing model checkers. We aim at investigating how the general approach of abstract interpretation can be applied to the context of counterexample-guided refinements of abstract models. In particular, we aim at generalizing the counterexample-guided refinement methodology to more general abstract models specified by abstract interpretation and to generic specification languages. We also plan to study the role of complete abstract domains in abstraction refinement algorithms.

- Task PD-4.2: Language Transformations
Strong preservation is now achieved by performing some refinement action of an abstract model. Our goal is to modify the specification language instead of the abstract model in order to achieve strong preservation. We plan to attack this kind of problem by using typical abstract interpretation techniques, in particular abstract domain refinements and simplifications. As a remarkable application, we plan to study whether this technique can be applied to the branching vs linear problem for the expressive power of temporal languages.

- Task PD-4.3: Algorithms for Computing Behavioural Equivalences
Most refinements of abstract models for achieving strong preservation w.r.t. a whole language boil down to computing some behavioural equivalence of the abstract model, notably bisimulation equivalence for the mu-calculus and CTL, simulation equivalence for ACTL* and branching bisimulation for CTL*-X. We recently designed a new simulation equivalence algorithm that improves the best known time bound for this problem. This procedure exploits typical abstract interpretation tools like abstract domain refinements and disjunctive abstract domains. We plan to investigate whether the techniques used for designing this new simulation equivalence may be generalized and adapted for other behavioural equivalences like branching bisimulation.*

The following Gantt chart shows the estimated project schedule of the above research tasks together with their dependency relationships assuming that the project will span the years 2008 and 2009.



14 - Ruolo di ciascuna unità operativa in funzione degli obiettivi previsti e relative modalità di integrazione e collaborazione

Testo italiano

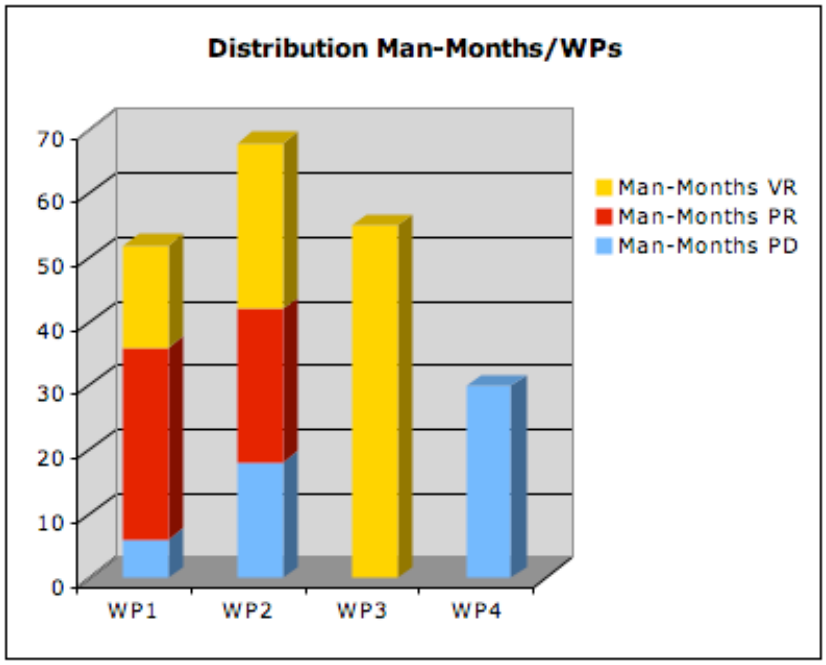
Come già discusso, il progetto AIDA2007 è organizzato in quattro WorkPackage (WP), con ogni workpackage a sua volta organizzato in diversi research task, i quali sono assegnati individualmente alle tre unità di ricerca: Padova (PD), Parma (PR) e Verona (VR). Il workpackage WP1 (Progettazione di interpretazioni astratte) tratta di quelle questioni del progetto di domini e semantiche astratte che sono relativamente indipendenti dalla particolare applicazione; gli altri tre workpackage sono diretti verso altrettante aree applicative. La distribuzione del carico di lavoro in termini di mesi-uomo per ogni unità di ricerca sui quattro workpackage è riportata nella seguente tabella:

	Man-Months PD	Man-Months PR	Man-Months VR
WP1	6	30	16
WP2	18	24	26
WP3	0	0	55
WP4	30	0	0

Si può osservare che, mentre tutte le unità di ricerca contribuiscono a WP1 e WP2, i workpackage WP3 e WP4 sono formati da task proposti da una singola unità (VR e PD, rispettivamente). I research task di WP3 (Protezione del codice mediante interpretazione astratta) e WP4 (Verifica di sistemi mediante interpretazione astratta) sono infatti più specializzati, mentre i temi e le questioni trattate in WP1 sono basate su un background teorico che è comune a tutti i membri di AIDA2007. Allo stesso modo, i research task elencati in WP2 (Analisi statica mediante interpretazione astratta) sono diretti, in un certo senso, verso le applicazioni più classiche dell'interpretazione astratta. Ci sono dunque, su WP1 e WP2, maggiori opportunità di raccogliere contributi da tutte le unità di ricerca.

La tabella mostra anche che il 75% di tutti i mesi-uomo del progetto sono dedicati a questioni relative all'applicazione di tecniche di interpretazione astratta: questa tendenza, che è comune alla maggior parte dei progetti di ricerca in questo settore, testimonia il livello di maturità del campo di ricerca.

Per una migliore visualizzazione della distribuzione dei mesi-uomo sui workpackage, i dati della tabella di cui sopra sono riportati sul seguente istogramma:



Si può notare che WP2 è il workpackage caratterizzato dal maggior carico di lavoro: non solo perché, come detto, WP2 raccoglie contributi da tutte le unità di ricerca, ma anche perché alcuni task di WP2 comportano del lavoro implementativo. Riguardo alla distribuzione del carico di lavoro tra le unità, si osservi che, naturalmente, Verona e Padova dedicano la maggior parte dei propri sforzi a WP3 e WP4, rispettivamente. D'altro canto, la maggior parte dei mesi-uomo di WP1 sono forniti dall'unità di Parma, che ha forti esperienze e competenze nel progetto, sviluppo e implementazione di domini e operatori semantici astratti. Infine, si vede come il carico di lavoro di WP2 sia distribuito in modo sostanzialmente equo tra le tre unità di ricerca.

Favoriremo le opportunità di un'esplicita e stretta collaborazione tra i ricercatori di AIDA2007, possibilmente favorendo e finanziando visite reciproche che portino a pubblicazioni congiunte. Inoltre, la collaborazione e l'integrazione tra unità di ricerca sarà incoraggiata e semplificata grazie all'organizzazione di riunioni di progetto, dove i risultati saranno presentati e discussi, così da raccogliere feedback prezioso ed ulteriori spunti per proseguire al meglio nel lavoro di ricerca. In particolare, contiamo di organizzare almeno tre riunioni, una per ogni unità di ricerca, dedicando l'ultima riunione alla presentazione e discussione dei risultati finali del progetto.

Si osservi come, oltre alle relazioni tra task affidati alla stessa unità di ricerca (che sono menzionate nei rispettivi programmi di ricerca) ci sono varie connessioni e/o dipendenze tra i task di unità di ricerca differenti: queste ultime saranno sfruttate per rafforzare la collaborazione tra le persone coinvolte nel progetto, sia con lavoro comune che attraverso lo scambio di idee e proposte. Queste connessioni sono elencate di seguito.

- C'è una forte correlazione tra le unità di Parma (Task PR-1.1) e Verona (Task VR-2.1) per ciò che concerne lo studio di domini astratti e analisi statiche che approssimano il "possible aliasing" in programmi Java. È interessante notare che i membri di Parma e Verona condividono un background teorico comune sull'analisi di sharing nei programmi logici, un tema che ha dimostrato la sua rilevanza per l'analisi di aliasing in Java.

- I task PR-2.4 e VR-2.1 sono fortemente correlati anche in ragione dell'approccio al problema dell'analisi di terminazione. Di nuovo, il lavoro di Parma e Verona può essere visto come originato da uno schema teorico comune, originato dal lavoro svolto sull'analisi di terminazione per i programmi logici, il quale viene poi specializzato in modo diverso: in particolare, il Task PR-2.4 si concentrerà sulle questioni peculiari all'analisi di terminazione di programmi C, mentre gli sforzi del Task VR-2.1 saranno concentrati sull'analisi di terminazione dei programmi Java.

- Parte del lavoro di ricerca nel Task VR-2.1 sarà dedicato allo studio di un'estensione corretta delle analisi di sharing, ciclicità e "path-length" in ambiente concorrente, nonché allo studio dell'analisi di "deadlock-freeness", la quale è rilevante per le prove di terminazione. Questo lavoro si intersecherà con lo sviluppo, nel Task PD-2.2, di un'analisi statica di proprietà causali per sistemi concorrenti, favorendo così il coordinamento e l'integrazione degli sforzi delle unità di Verona e Padova.

- Infine, c'è un'interessante dipendenza tra i Task VR-1.1 e PD-4.2. Il problema di ricerca teorica che è comune ad entrambi è lo studio di trasformazioni della semantica che siano in grado di ottenere la completezza rispetto ad un'astrazione fissata: questo è un approccio complementare rispetto a lavori precedenti, nei quali la semantica era fissata e la completezza doveva essere ottenuta trasformando il dominio astratto. La soluzione del problema generale, formalizzato in modo indipendente dal dominio, perseguita nel contesto del Task VR-1.1, guiderà lo sviluppo di (e riceverà feedback da) una sua istanza specifica, ovvero dipendente dall'applicazione considerata del task PD-4.3, dove l'accento è sulle trasformazioni di linguaggi per il model checking astratto "strongly preserving". Vale la pena richiamare qui che gli stessi risultati saranno sfruttati, dall'unità di Verona, anche nel contesto di WP3 per applicazioni legate al software watermarking.

Oltre alle forme classiche di collaborazione summenzionate, crediamo che, per un'integrazione più pratica dei risultati e degli approcci proposti dalle unità di ricerca partecipanti al progetto, un ruolo importante sia giocato dallo sviluppo di strumenti software come, ad esempio, la PPL (Parma Polyhedra Library), distribuita nei termini della GNU General Public License (GPL), il che semplifica e incoraggia l'integrazione delle idee proteggendone al contempo la proprietà intellettuale. Questo tipo di strumenti consente la disseminazione rapida di ogni avanzamento concettuale e tecnologico, non solo tra le unità che partecipano al progetto, ma anche a gruppi internazionali di ricerca che operano nel campo dell'interpretazione astratta, aumentando così la risonanza complessiva dei risultati ottenuti. Il passato recente testimonia qualche positiva esperienza in tal senso, tra le quali la prototipazione rapida di un'analizzatore statico sviluppato da ricercatori dell'unità di Verona basato, tra l'altro, su domini astratti e tecniche di analisi messe a disposizione dalla PPL.

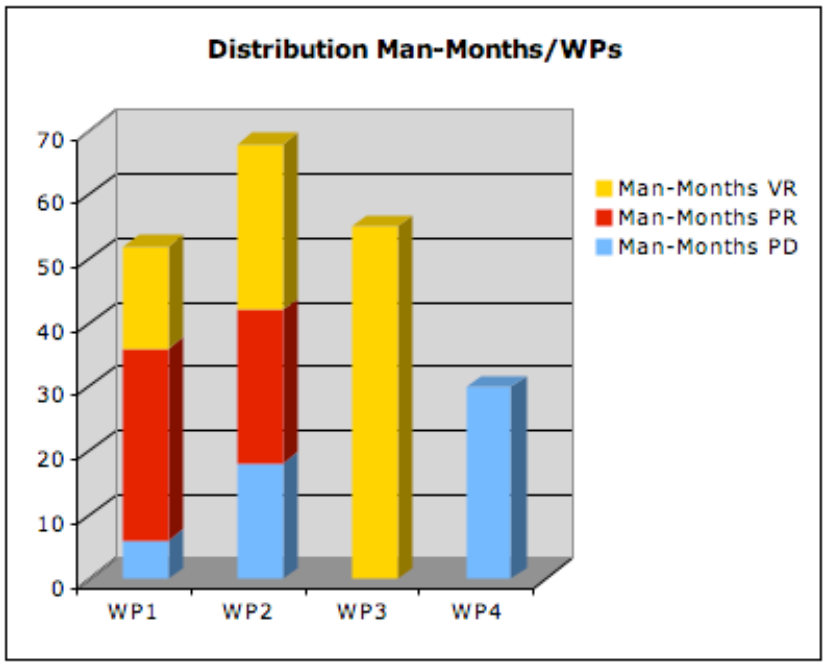
Testo inglese
As discussed above, the AIDA2007 project is organized in four WorkPackages (WPs), each workpackage being itself organized in several research tasks, which are individually contributed by the three research units Padova (PD), Parma (PR) and Verona (VR). The workpackage WP1 (Abstract Interpretation Design) deals with those issues in the design of abstract domains and abstract semantics that are somehow independent from the particular application at hand; the other three workpackages target specific application areas. The distribution of the workload in terms of man-months of each research unit on the four workpackages is reported in the following table:

	Man-Months PD	Man-Months PR	Man-Months VR
WP1	6	30	16
WP2	18	24	26
WP3	0	0	55
WP4	30	0	0

It can be observed that all the research units contribute to WP1 and WP2, whereas WP3 and WP4 are each formed by tasks proposed by a single research unit (Verona and Padova, respectively). This is because the research tasks listed in WP3 (Code Protection by Abstract Interpretation) and WP4 (System Verification by Abstract Interpretation) are more specialized. In contrast, the topic and issues addressed in WP1 are based on a theoretical background that is common to all the members of AIDA2007. Similarly, the research tasks listed in WP2 (Static Analysis by Abstract Interpretation) can be seen as targeting the more classical applications of abstract interpretation. Hence, for both WP1 and WP2 there are more opportunities of contributions coming from all of the research units.

The table also shows that as much as 75% of all the man-months of the project are devoted to the investigation of application-specific issues: this is a trend which is common to most research projects in this area, witnessing the maturity level of the research field.

For a better illustration of the relative amount of man-months devoted to each workpackage, the same data of the table above can be cast in the following histogram:



Here, it can be seen that WP2 is the workpackage having the higher workload: this is justified not only for the reason already mentioned above (WP2 collects contributions from all research units), but also from the fact that some of the tasks composing WP2 include some implementation work. Regarding the relative distribution of the units' workloads, it can be observed that, as expected, Verona and Padova devote most of their efforts to workpackages WP3 and WP4, respectively. On the other hand, most of the man-months of WP1 are provided by the unit of Parma, which has strong traditions and skills in the design, development and implementation of abstract domains and abstract semantics operators. Finally, the workload for WP2 can be seen to be almost evenly distributed between the three research units.

We will encourage the possibility of an explicit, tight collaboration between the researchers in AIDA2007, possibly by reciprocal visits of members leading to joint publications. Furthermore, collaboration and integration between research units will be encouraged and made easier by the organization of several project meetings, where intermediate results will be presented and discussed, so as to provide useful feedback and further hints on how to better proceed in the research work. In particular, we plan to organize at least three meetings, one for each research unit, with the last one devoted to the presentation and discussion of the final results of the project.

Note that, besides the interrelations between tasks belonging to the same research unit, that are mentioned in the corresponding research programs, there are also several interrelations and/or dependencies between tasks of different research units, that will be suitably exploited to strengthen the collaboration between people involved in the project, either by joint work or by exchanging ideas and proposals. These interrelations are listed below.

- There is a strong correlation between the units of Parma (task PR-1.1) and Verona (task VR-2.1) for what concerns the study of abstract domains and static analysis tools that track possible aliasing in Java programs. It should be noted that the members of Parma and Verona share a common theoretical background in the analysis of sharing properties of logic programs, which has shown to be an influential topic for the subject.

- Tasks PR-2.4 and VR-2.1 are also strongly related for what concerns the overall approach with respect to the problem of termination analysis. In this case too, the work of Parma and Verona can be seen as starting from a common theoretical setting, borrowed from previous work on termination analysis for logic programs, which is then specialized differently: in particular, Task PR-2.4 will target the issues that are peculiar to termination analysis of C programs, whereas the efforts in Task VR-2.1 will be focused on termination analysis for Java programs.

- Part of the research work in Task VR-2.1 will be devoted to the study of a correct extension of the analyses of sharing, cyclicity and path-length in the presence of concurrency, as well as on the analysis of deadlock-freeness, which is relevant for termination proofs. This is going to have some intersection with the development of a static analysis of causal properties for concurrent systems, which will be studied in Task PD-2.2, thereby favoring the coordination and integration of the efforts of

the units of Verona and Padova.

- Finally, there is an interesting dependence between Tasks VR-1.1 and PD-4.2. The theoretical research problem which is common to both tasks is the design of suitable transformations of the semantics that are able to achieve completeness with respect to a fixed abstraction: this is a complementary approach with respect to similar, previous work where the semantics was fixed and completeness had to be achieved by transforming the abstract domain. The solution of the application-independent formulation of the problem, pursued in the context of Task VR-1.1, will drive the development of (and will receive feedback from) its application-dependent instantiation in Task PD-4.3, where the focus is on language transformations for strongly-preserving abstract model checking. It is also worth recalling here that the same results will be also exploited, by the unit of Verona, in the context of WP3, for applications related to software watermarking.

Besides the classical forms of collaborations mentioned above, we believe that, for a more practical integration of the results and approaches put forward by the research units participating to the project, an important role can be played by the development of software tools such as, for instance, the PPL (Parma Polyhedra Library), distributed under the GNU General Public License (GPL), which simplifies and encourages the integration of ideas while properly protecting intellectual property. This kind of tools allow for a quick dissemination of any conceptual and technological advancement, not only between the units participating to the project, but also to other, international research groups operating in the field of abstract interpretation, thereby enhancing the overall impact of the results obtained. The recent past witnesses some positive experiences in this respect, among which the rapid prototyping of a static analysis tool developed by people of the research unit in Verona that was also based on abstract domains and techniques made available by the PPL.

15 - Risultati attesi dalla ricerca, il loro interesse per l'avanzamento della conoscenza e le eventuali potenzialità applicative

Testo italiano

L'intento della proposta di progetto AIDA2007 è l'avanzamento dello stato dell'arte sull'interpretazione astratta (IA), sia dal punto di vista fondazionale, che da quello applicativo. Ciò avverrà tramite lo studio di specifici "Task" all'interno dei quattro WP che costituiscono AIDA2007. Vengono descritti nel seguito i risultati attesi da ogni singolo Task.

WP1: Progettazione di Interpretazioni Astratte

- Task PD-1.1: Domini astratti per il model checking
I domini astratti disgiuntivi risultano molto utili nel model checking astratto, e.g. nella predicate abstraction [MFHRS06] e negli algoritmi per il calcolo di equivalenze comportamentali [RT07b]. Intendiamo studiare come questi domini possano essere usati negli algoritmi di abstraction refinement. Poiché i domini astratti disgiuntivi sono caratterizzabili in termini di domini astratti completi per l'unione di insiemi, ci attendiamo di caratterizzare anche classi di domini astratti completi che possano essere vantaggiose per l'abstract model checking, e.g. domini astratti completi per qualche operatore temporale.

- Task PR-1.1: Domini astratti per l'analisi dei puntatori e dell'aliasing in programmi C e Java
L'approccio di [EGH94] alla stack pointer analysis di programmi C permette di definire un'analisi dipendente dal contesto e/o dal flusso capace di catturare l'aliasing sia potenziale che effettivo. La specifica in [EGH94] è piuttosto informale e rende quindi difficile un ragionamento rigoroso sulla correttezza e precisione dell'approccio proposto. Intendiamo riprogettare, formalizzare e implementare una variante di quel dominio, considerando in particolare come semantica concreta di riferimento quella specificata dallo standard del linguaggio C.
Per l'analisi di aliasing di programmi Java, individueremo un framework concettuale comune che inquadri le proposte che si trovano in letteratura. Riformuleremo le varianti più note di analisi di aliasing e di shape come astrazioni di una stessa semantica denotazionle, sensibile al flusso di esecuzione a al contesto. Ci aspettiamo di riuscire a confrontare formalmente i vari approcci, e di distinguere la rappresentazione tramite domini astratti delle strutture dati dinamiche dagli algoritmi per il calcolo del comportamento astratto.

- Task PR-1.2: Domini astratti scalabili a precisi
Esiste un'ampia letteratura sui domini astratti numerici: da semplici proprietà non relazionali a domini che codificano proprietà relazionali complesse. La questione è come avvicinarsi all'efficienza e scalabilità dei domini più semplici ottenendo una precisione simile a quella dei domini più complessi. Gli studi attuali esplorano lo spazio esistente tra gli estremi sopra menzionati cercando dei compromessi. In questo Task intendiamo progettare (1) operatori approssimati sui domini esistenti e (2) nuove combinazioni di domini esistenti e dei corrispondenti operatori per integrarli efficacemente.

- Task VR-1.1: Trasformazioni semantiche orientate alla completezza.
Ci aspettiamo come risultati: (1) La chiusura del problema della completezza in IA, affiancando i trasformatori di semantiche ai ben noti trasformatori di domini astratti; (2) Poiché trasformare semantiche significa trasformare programmi, l'aver reso completa una semantica implica che il corrispondente programma è stato sottoposto ad una deformazione la cui IA è completa; (3) Il problema duale di rendere la semantica il più possibile incompleta corrisponderà a massimizzare la capacità del programma deformato ad includere watermark.

- Task VR-1.2: Misura di precisione basata su domini numerici.
Studieremo la struttura del reticolo delle IA probabilistiche ereditata dal reticolo delle proiezioni di Birkhoff-von Neumann, una misura della precisione relativa calcolata mediante un operatore di norma associato, e un'interpretazione statistica del valore che esprime la precisione relativa. Un aspetto importante sarà la rappresentazione della semantica di un programma come prodotto tensore degli spazi vettoriali associati al dominio di ogni sua variabile; con ciò collegheremo la nozione di dipendenza relazionale nell'ambito dell'analisi di programmi con quelle statistiche di correlazione e indipendenza.

WP2: Analisi Statica mediante Interpretazione Astratta

- Task PD-2.1: Semantiche causali di sistemi concorrenti e distribuiti
Abbiamo sviluppato in [CVY07] una semantica composizionale basata su strutture ad eventi per il pi-calculus interno che cattura le dipendenze causali generate dal "name passing" sia sincrono che asincrono. Estenderemo questo approccio al "free name passing" per ottenere una semantica denotazionale fully abstract per il full pi-calculus. Studieremo inoltre semantiche composizionali simili per linguaggi imperativi multithreaded che, rispetto alle algebre di processi, si focalizzano sullo stato piuttosto che sul controllo di un thread.

- Task PD-2.2: Analisi statica di proprietà causali
La letteratura propone diverse nozioni di equivalenza su strutture ad eventi. Ci aspettiamo di individuare caratterizzazioni logiche di alcune di queste equivalenze, indagando in particolare sulle relazioni tra l'osservazione di distribuzione esprimibile mediante logiche spaziali per il pi-calculus e quella esprimibile tramite strutture ad eventi.
Il problema del model checking su strutture ad eventi è intrinsecamente ostico, poiché in generale queste strutture sono modelli infiniti anche per processi con un numero finito di stati. Ci aspettiamo che l'approccio dell'IA al model checking possa essere applicato con successo all'abstract model checking di strutture ad eventi.

- Task PR-2.1: Definizione di uno schema di analisi modulare e generico
Studieremo i problemi collegati al progetto e all'implementazione di uno schema generico di analizzatori statici per i linguaggi imperativi di largo uso, che sarà istanziato su analisi dei puntatori e dei valori delle variabili numeriche e booleane. Le innovazioni principali di questo schema saranno la flessibilità e la generalità, sia in termini delle proprietà che in termini del linguaggio analizzato.

- Task PR-2.2: Annotazione di programmi
Studieremo la specifica di un linguaggio di asserzioni che consenta al programmatore di codificare nel programma le condizioni logiche che garantiscono che lo stesso sia immune da una certa classe di errori e/o rispetti i limiti imposti dalle politiche di allocazione delle risorse. La presenza di un linguaggio di asserzioni espressivo sarà sfruttata anche per abilitare la comunicazione dall'analizzatore al programmatore. Il risutato atteso è un ambiente di sviluppo fortemente integrato,

dove il feedback derivante dal confronto delle proprietà attese e quelle calcolate permetterà un processo di raffinamento sinergico, con benefici sia per lo sviluppo che per il debugging delle applicazioni.

- Task PR-2.3: Analisi di correttezza della manipolazione di stringhe
Studieremo un approccio trasformazionale al problema di assicurare la correttezza delle operazioni sulle stringhe nei programmi C. L'approccio consiste nel trasformare un programma P che usa stringhe, in un'altro, P', dove gli array di caratteri e i puntatori sono sostituiti da strutture ad hoc e dove sono state inserite delle asserzioni. La trasformazione (1) sarà completamente automatica e (2) tale che, se P' non può violare nessuna delle sue asserzioni, allora ogni esecuzione di P sarà esente da errori nella manipolazione delle stringhe. Le stringhe saranno sostituite da strutture che astraggono la loro semantica per mezzo di variabili intere e booleane. L'idea nasce dai lavori di Dor, Ellenbogen, Rodeh e Sagiv [DRS01, DRS03].

- Task PR-2.4: Analisi di terminazione e complessità
Ci attendiamo di progettare un'analisi di terminazione e complessità per il codice C nel modo seguente:
1) Identificazione dei cicli nel controllo: Si identificano le sorgenti di possibile non terminazione: cicli espliciti, chiamate di funzione ricorsive e salti all'indietro.
2) Identificazione dei cicli nei dati: sfruttando l'analisi dei puntatori viene analizzata la ciclicità delle strutture dati. Questa informazione servirà nel passo 3.
3) Generazione di vincoli: a) sui valori delle variabili numeriche; b) sulla lunghezza massimale di un percorso di puntatori seguito a partire da ciascuna variabile (path-length analysis).
4) Sintesi di funzioni di ranking: i vincoli generati vengono elaborati con tecniche automatiche che cercano di identificare una funzione di terminazione (norma) per il programma.

- Task VR-2.1: Analisi statica di terminazione e complessità
Ci aspettiamo di migliorare ed estendere i nostri tool Julia e BinTerm. Attulamente, la nostra analisi di terminazione basata su Julia e BinTerm è in grado di provare la terminazione di piccoli programmi Java bytecode, anche con strutture dati. L'analisi sarà estesa a programmi realistici. Per questo intendiamo migliorare le analisi preliminari rendendole più veloci e precise. Ci aspettiamo di incrementare l'efficienza della Parma Polyhedra Library considerando la forma speciale dei nostri poliedri, dove molte variabili di input e output mantengono lo stesso valore. Ci aspettiamo di sviluppare un'analisi statica di sharing, ciclicità e path-length nel caso di programmi Java concorrenti, e di integrare l'esistente analisi di terminazione con dimostrazioni automatiche di assenza di deadlock.

- Task VR-2.2: Ottimizzazione speculativa
La parallelizzazione del codice rappresenta un problema difficile: quando un compilatore deve parallelizzare due pezzi di codice deve considerare tutte le possibili dipendenze. Le tecniche esistenti sono "over-conservative": se una dipendenza non può essere provata, si assume che ci sia. Quindi molte opportunità di parallelizzazione vengono scartate. Contrariamente a questo approccio conservativo di "caso pessimo", il threading speculativo è una tecnica emergente che permette alcune parallelizzazioni incorrette di thread ritardando la verifica ad un momento successivo. Ci aspettiamo di applicare l'IA probabilistica per supportare questo approccio "ottimistico".

WP3: Protezione del Codice mediante Interpretazione Astratta

- Task VR-3.1: Generalizzazione di firme
Ci aspettiamo di sviluppare un metodo di generalizzazione della firma che riconosca anche le varianti metamorfiche. Consideriamo un EFSA (Extended Finite State Automaton) S che modella la firma e un possibile offuscamento O utilizzato da un hacker. Se il modello dell'EFSA è chiuso rispetto ad O, allora la generalizzazione dovrebbe restituire l'insieme di tutti i possibili EFSA che possono essere ottenuti da S attraverso O, e questo può essere usato per identificare le varianti metamorfiche originate da O. La classe di trasformazioni per cui EFSA è chiuso corrisponde alla classe di offuscamenti che la generalizzazione può gestire.

- Task VR-3.2: Predicate obfuscation
Siamo interessati alla possibilità di nascondere una specifica proprietà di un particolare programma (o classe di programmi). Vogliamo studiare l'esistenza di trasformazioni di codice, dette predicate-obfuscation, che cercano di proteggere uno specifico tipo di informazione di un certo programma (o classe di programmi). Ci aspettiamo quindi di caratterizzare la classe di predicati che è possibile nascondere in modo da precisare le potenzialità e i limiti dell'offuscamento.

- Task VR-3.3: Inserimento di software watermark in cicli
Intendiamo inserire, all'interno di un ciclo, dei watermark calcolati iterativamente. Ciò si attua dispiegando interamente il ciclo e cercando l'iterazione più adeguata per l'inserimento del watermark, detta iterazione promotrice. Collegando poi l'inizializzazione del watermark a tale iterazione e inserendo il watermark così modificato nel ciclo originale, renderemo difficile l'individuazione dell'iterazione promotrice. L'estrazione sarà guidata da una chiave che useremo per dipanare soltanto l'iterazione promotrice e quelle ulteriori richieste per la computazione del watermark.

- Task VR-3.4: Nascondere watermark in lacune di completezza
Offuscamento e incompletezza sono concetti profondamente legati: mentre la steganografia può essere vista come il nascondere informazione in lacune di completezza, l'offuscamento può essere visto come il trasformare programmi in modo che le uniche astrazioni complete utilizzabili dagli osservatori siano molto costose, ad esempio analisi relazionali o esponenziali sul numero di variabili. Basandoci su questa idea, ci attendiamo di sviluppare una nuova famiglia di metodi e strumenti perla stenografia che siano ben definiti e dimostrabilmente sicuri, e piu` in generale per l'offuscamento. Ci aspettiamo di mostrare che molte tecniche di software watermarking sono casi speciali di inserimento di watermark basato sull'incompletezza dell'osservatore. Siamo interessati allo studio di attacchi passivi e attivi e utilizzeremo i risultati del Task VR-1.1 per fornire un modello per questi comportamenti.

WP4: Verifica di Sistemi mediante Interpretazione Astratta

- Task PD-4.1: Algoritmi di raffinamento di astrazioni
I model checker basati sulla tecnica dei raffinamenti di astrazioni guidati da controesempi (CEGAR, dall'acronimo inglese) considerano come controesempi dei cammini di stati astratti. Recentemente, Cousot, Ganty e Raskin [CGR07] hanno introdotto un algoritmo di raffinamento di astrazioni CGR che è guidato da punti fissi astratti piuttosto che da controesempi astratti. Intendiamo studiare se e come l'approccio generale dell'IA possa essere applicato in questo contesto. Ci aspettiamo di generalizzare la metodologia CEGAR a generici modelli astratti specificati tramite IA ed a generici linguaggi di specifica. Intendiamo inoltre studiare il ruolo dei domini astratti completi negli algoritmi di raffinamento di astrazioni. Ci aspettiamo che i raffinamenti di completezza possano essere utili per progettare o migliorare gli algoritmi di raffinamento di astrazioni, in particolare nel caso degli algoritmi guidati da punti fissi come l'algoritmo CGR.

- Task PD-4.2: Trasformazioni di linguaggi
Dato un modello astratto A, intendiamo studiare il problema di modificare sistematicamente, attraverso raffinamenti o semplificazioni minimali, gli operatori che definiscono qualche linguaggio L (e.g. OR e AU) con lo scopo di ottenere un linguaggio trasformato L' che sia tale che l'abstract model checking su A sia strongly preserving per L'. A questo scopo useremo tipiche tecniche di IA, in particolare raffinamenti e semplificazioni di domini astratti. Intendiamo applicare tale metodo al problema "branching vs linear" nel potere espressivo dei linguaggi temporali. Generalizzando la caratterizzazione in [Mai00], ci aspettiamo di trasformare sistematicamente LTL in un sotto-linguaggio massimale L' che caratterizza il potere espressivo branching di ACTL.

- Task PD-4.3: Algoritmi di calcolo di equivalenze comportamentali
Abbiamo recentemente individuato un nuovo algoritmo di calcolo dell'equivalenza di simulazione [RT07b] che migliora la complessità di tempo sinora conosciuta. Questa procedura si appoggia su tipici strumenti dell'IA come i raffinamenti di domini astratti e i domini astratti disgiuntivi. Una valutazione sperimentale ha inoltre dimostrato l'effettività pratica di questo algoritmo. Ci aspettiamo: (1) una versione simbolica efficiente di questo algoritmo, basata su BDDs e (2) una generalizzazione delle tecniche usate per questo nuovo algoritmo che permetta l'adattamento ad altre equivalenze comportamentali come la branching bisimulation.

Testo inglese
The overall goal of the AIDA2007 project proposal is to advance the state-of-the-art of research on abstract interpretation, both on foundational and applicative levels. This will be carried out by investigating some specific tasks within the four WPs that constitute AIDA2007. The following description provides a summary of the expected results for each single task.

WP1: Abstract Interpretation Design

- Task PD-1.1: Abstract domains for model checking
Disjunctive abstract domains turn out to be particularly useful in abstract model checking, e.g. in predicate abstraction [MFHRS06] and in algorithms for computing behavioural equivalences [RT07b]. We plan to investigate how disjunctive abstract domains can be useful in abstraction refinement algorithms. Since disjunctive abstract domains can be characterized as abstract domains that are complete for set unions, we also expect to characterize classes of complete abstract domains that can be advantageously used in abstract model checking, e.g. abstract domains that are complete for some temporal operator.

- Task PR-1.1: Abstract domains for points-to and aliasing analysis of C and Java programs
The approach put forward in [EGH94] for stack pointer analysis of C programs allows for the definition of a flow- and context-sensitive analysis that is able to capture both possible and definite aliasing. The specification in [EGH94] is rather informal and, hence, a precise reasoning on the correctness and precision of the approach is difficult. We plan to re-design, properly formalize and implement a variant of this domain, in particular considering as reference concrete semantics the one specified by the C language standard.
For aliasing analysis in Java programs, we will identify a common conceptual framework where most of the proposals in the literature may be cast. We plan to reformulate the most common variants of aliasing and shape analysis as abstractions of a common denotational, flow and context sensitive semantics. We expect, as a result, the ability to formally evaluate the relative strength and weakness of the various approaches, as well as the ability to decouple the abstract domains representing the dynamic data structures from the algorithms computing the abstract behavior.

- Task PR-1.2: Scalable and precise numeric abstract domains
There is a vast body of literature concerning the design of numeric abstract domains, ranging from simple non-relational properties to sophisticated relational properties. Clearly, issues arise when the efficiency and scalability of the simpler proposals has to be paired with the precision of the more sophisticated ones, so that most of the current research, looking for appropriate tradeoffs, lies in between of the two extremes mentioned above. We will attack this problem by (1) designing approximated operators on the existing domains and (2) designing new combinations of existing domains and of the corresponding operators for their effective integration.

Task VR-1.1: Completeness-driven semantic transformations
We expect the following results: (1) To complete the study of the key problem of completeness in abstract interpretation by adding semantic transformers to the already known abstract domain transformers; (2) Transforming semantics means transforming programs [CC02]. This means that with each transformed complete semantics we will have a corresponding program deformation whose abstract interpretation is complete. (3) The dual problem of making semantics maximally incomplete will correspond to let the deformed code be maximally capacious for holding watermarks.

- Task VR-1.2: Measuring precision in numerical domains
We will investigate the lattice structure of probabilistic abstract interpretations inherited by the Birkhoff-von Neumann lattice of projections, a measure of the relative precision via an associated operator norm and the statistical interpretation of the number expressing the relative precision. An important point in this investigation is the representation of the semantics of a program via a tensor product of the vector spaces associated to the domain of each program variable. This establishes a connection between the notion of relational dependencies in program analysis and that of correlation and independence in statistics.

WP2: Static Analysis by Abstract Interpretation

- Task PD-2.1: Causal semantics for concurrent and distributed systems
We developed in [CVY07] a compositional event structure semantics for the internal pi-calculus, that captures the essential features of the causal dependencies created by both synchronous and asynchronous name passing. We plan to extend this approach to free name passing, and we expect to obtain a fully abstract denotational semantics for the full pi-calculus. We are also interested in studying similar compositional semantics for multithreaded imperative languages, that, compared to process algebras, focus on the state rather than on the control of a thread.

- Task PD-2.2: Static analysis of causal properties
In the literature a number of equivalence notions over event structures exist. We expect to provide logical characterizations of the equivalences over event structures, in particular by investigating the connection between the distribution observation expressed in the spatial logics for the pi-calculus and the one expressible through event structures.
The model checking problem over event structures is an intrinsically difficult task, since in general event structures are infinite models even for finite state processes. We expect that the abstract interpretation approach can be fruitfully applied for defining an abstract model checking framework over event structures.

- Task PR-2.1: Definition of a modular and generic analysis framework
We will study the problems related to the design and implementation of a generic framework for static analysis of imperative languages in widespread use, which will be later instantiated to obtain analyses to approximate pointer aliasing and the values of numeric and Boolean variables. The main innovations of such a framework will be flexibility and generality, both in terms of the analyzed properties and of the analyzed language.

- Task PR-2.2: Annotation of programs
We will investigate the specification of a suitable assertion language that should enable the programmer to encode in the program itself the logic conditions that guarantee that the program is unaffected by some given class of errors and/or it respects the bounds imposed by the resource allocation policies. The availability of an expressive assertion language will also be exploited to enable communication the other way around, from the analyzer to the programmer. The expected final result is a highly integrated development environment, where feedback coming from the comparison of expected and computed properties can start a synergistic refinement process, benefiting both the application development and debugging phases.

- Task PR-2.3: Analysis of string cleanness
We will study a transformational approach to the problem of ensuring cleanness of string operations in C programs. The approach consists in transforming a C program P using strings into another program, P', where arrays of characters and pointers are substituted by ad hoc structures and where assertions are included. The transformation will have the following properties: (1) it will be fully automatic; (2) if P' cannot violate any of its assertions, then all the executions of P do not result in any string manipulation errors. Strings will be replaced by structures that abstract their semantics by means of integer variables and Boolean variables. This idea originates from the works of Dor, Ellenbogen, Rodeh and Sagiv [DRS01, DRS03].

- Task PR-2.4: Analysis of termination and complexity
We expect to design a termination and complexity analysis for C code as follows:
1) Identify control loops: All the sources of possible non-termination in a C program are identified: explicit loops, recursive function calls and backwards jumps.
2) Identify loops in data: By exploiting the information provided by the pointer analysis, possible loops in data structures are examined. This information is used in step 3 below.
3) Generation of constraints: a) on the values of numeric variables; b) on the maximal length of a chain of pointers followed starting from each variable (path-length analysis).
4) Synthesis of ranking functions: The generated constraints are elaborated through automatic techniques so as to try and find a termination function (norm) for the program.

- Task VR-2.1: Static analysis for termination ad complexity
We expect to improve and extend our Julia and BinTerm tools in a number of different ways. Our current termination analysis based on Julia and Binterm is already able to prove termination of small Java bytecode programs, also dealing with data structures. We plan to extend it to the analysis of large, real programs. To that purpose, preliminary analyses will be improved so that they become both faster and more precise. We expect to improve the efficiency of the Parma Polyhedra Library for our purposes by considering the special shape of our polyhedra, where most input and output variables keep the same value. We expect to develop a static analysis of sharing, cyclicity and path-length for concurrent Java programs. We also expect to integrate in our current termination analysis automatic proofs of deadlock-freeness.

- Task VR-2.2: Speculative optimisation

Code parallelisation represents one of today's major challenges. When a compiler has to parallelise two pieces of code it has to consider all potential dependencies. Current techniques are over-conservative: if a dependency cannot be proved, the compiler assumes that it does occur. As a consequence, many opportunities for parallelisation are missed in the code generation. Contrary to this "worst-case" conservative approach, speculative threading is an emerging technique which allows for some incorrect thread parallelisation by postponing the check to some later time. We expect to apply probabilistic abstract interpretation to support such "optimistic" approach.

WP3: Code Protection by Abstract Interpretation

- Task VR-3.1: Signature generalization
We expect to develop a signature generalization process that detects metamorphic malware variants. Let S be a Extended Finite State Automata (EFSA) modeling the malware signature and let O be an obfuscation used by malware writers. If the EFSA model is closed for transformation O then the generalization process returns Closure(S,O) that can be used to detect all possible variants of S obtained through O. The class of transformations for which EFSA is closed corresponds to the class of obfuscations that our generalization can handle.

- Task VR-3.2: Predicate obfuscation
We are interested in investigating the possibility of hiding a precise predicate of a particular program (or class of programs). We expect to define obfuscation of predicates, i.e. the program transformations that aim at protecting a particular information about a program (or a class of programs). We aim at characterizing the predicates that is possible to hide in order to provide a more useful description of the potentiality and limits of code obfuscation.

- Task VR-3.3: Hiding software watermarks in loop structures
Our goal is to inlay inside loops watermarks that are iteratively constructed. At embedding time we plan to unfold the subject loop to find out which iteration is the most appropriate for promoting the construction of our watermark. Then we make our watermark sensible only to the promoter and we can embed it in the folded loop. Since the marked loop is folded, one can hardly guess which iteration should be the promoter. At extraction time, we plan to use a semantic understanding of loop unfolding together with an extraction key to display just the promoter followed by as many iterations as needed to compute our watermark.

- Task VR-3.4: Hiding watermarks in completeness holes
*Consider the following simple statement C: x=a*b multiplying a and b and storing the result in x. A (passive) observer inspecting the sign of program values has a complete understanding of the sign of the result by knowing the sign of a and b. An automated program sign analysis that replaces concrete computations with approximated ones -- i.e. the rule of signs -- is therefore able to catch, with no loss of precision, the intended sign behaviour of C because the sign abstraction is complete for integer multiplication. A transformation t replacing C with t(C): x=b; while b !=0 {x = a+x; b = b-1;} obfuscates the sign information for the observer. This because the rule of signs is incomplete for integer addition as signs loose integer magnitude. Thus, while steganography can be seen as hiding information in completeness holes, code obfuscation can be analogously seen as transforming programs in such a way that possible observers cannot use complete abstractions for code analysis unless cost expensive analysis are used. We expect to show that most software watermarking techniques are special cases of watermark insertions that exploit incompleteness of defined observers.*

WP4: System Verification by Abstract Interpretation

- Task PD-4.1: Abstraction Refinement Algorithms
Model checkers based on CounterExample-Guided Abstraction Refinement (CEGAR) deal with counterexamples that are paths of abstract states. Recently, Cousot, Ganty and Raskin [CGR07] put forward an innovative abstraction refinement algorithm CGR that is driven by abstract fixpoints rather than by abstract counterexamples. We aim at investigating how the general approach of abstract interpretation can be applied to the context of counterexample-guided refinements of abstract models. We expect to generalize the CEGAR approach to general abstract models specified by abstract interpretation and to generic languages. We also plan to study the role of complete abstract domains in abstraction refinement algorithms. We expect that completeness refinements can be useful for designing or improving abstraction refinement algorithms, in particular in the context of fixpoint-guided algorithms like CGR.

- Task PD-4.2: Language Transformations
Given an abstract model A, we plan to study the problem of systematically modifying, through minimal refinements or simplifications, the operators that define some specification language L (e.g. OR and AU) in order to obtain a transformed language L' such that the abstract model checking on A is strongly preserving for L'. We plan to attack this kind of problems by using typical abstract interpretation techniques, in particular abstract domain refinements and simplifications. We intend to apply this technique to the branching vs linear problem for the expressive power of temporal languages. By generalizing the characterization in [Mai00], we expect to systematically transform LTL to a maximal sub-language L' that characterizes the branching expressive power of ACTL.

- Task PD-4.3: Algorithms for Computing Behavioural Equivalences
We recently designed a new simulation equivalence algorithm [RT07b] that improves the best known time bound for this problem. This procedure exploits typical abstract interpretation tools like abstract domain refinements and disjunctive abstract domains. An experimental evaluation has shown the effectiveness of this algorithm. We expect (1) to provide an efficient symbolic version based on BDDs of this algorithm and (2) to generalize and adapt the techniques used for designing this new simulation equivalence algorithm to other behavioural equivalences like branching bisimulation.

16 - Elementi e criteri proposti per la verifica dei risultati raggiunti

Testo italiano
La valutazione globale dei risultati del progetto dovrebbe verificare che i principali obiettivi di AIDA2007 siano stati raggiunti. Ciò significa l'avanzamento dello state dell'arte nelle seguenti aree dell'interpretazione astratta:

WP1: Progettazione di interpretazioni astratte

- domini astratti disgiuntivi per il model checking e il raffinamento di astrazioni
- domini astratti scalabili per l'analisi di puntatori e variabili numeriche
- semantiche astratte complete e trasformazioni semantiche guidate dalla completezza
- misure quantitative della precisione dei domini astratti

WP2: Analisi statica mediante interpretazione astratta

- analisi di proprietà causali di sistemi concorrenti e distribuiti
- analisi di terminazione e complessità per programmi C e Java
- analisi di correttezza della manipolazione di stringhe per programmi C
- framework generici e modulari in grado di gestire programmi annotati
- ottimizzazione speculativa tramite interpretazione astratta probabilistica

WP3: Protezione del codice mediante interpretazione astratta

- riconoscimento di varianti offuscate della firma dei malware
- offuscamento del codice per specifici predicati sensibili

- inserimento di watermark software nei cicli e nelle lacune di completezza

WP4: Verifica di sistemi tramite interpretazione astratta

- raffinamento di astrazioni guidato da controesempi tramite interpretazione astratta
- trasformazione di linguaggi temporali allo scopo di ottenere la strong preservation
- algoritmi basati su interpretazione astratta per calcolare equivalenze comportamentali

La verifica del raggiungimento di questi obiettivi dovrà essere basata principalmente sugli articoli di ricerca che saranno presentati a conferenze e workshop internazionali con revisore o che saranno pubblicati su riviste internazionali.

Ci aspettiamo anche lo sviluppo di nuovi strumenti software e il miglioramento di quelli esistenti, come PPL, Julia e BinTerm. In particolare, ci aspettiamo di implementare e valutare l'efficacia dei domini astratti e delle analisi statiche sviluppate all'interno di WP1 e WP2. Tutti questi strumenti saranno resi disponibili a tutti e, quando possibile, saranno rilasciati con una licenza GNU GPL.

Uno degli scopi principali di AIDA2007 è di mobilitare, unificare e stimolare la comunità italiana dei ricercatori in interpretazione astratta. In particolare, il nostro obiettivo è promuovere la collaborazione tra membri di differenti unità di ricerca, specialmente per ciò che riguarda i WP1 e WP2. Il successo di questa collaborazione potrà essere misurato in termini di articoli e contributi scritti in maniera congiunta, scambi di benchmark, strumenti e librerie software, e dall'organizzazione di riunioni di progetto.

Allo scopo di facilitare il processo di valutazione, forniremo un sito web dove saranno raccolti tutti i prodotti della ricerca collegati ad AIDA2007 (articoli, strumenti software, benchmark, etc.).

Testo inglese
The overall evaluation of the project's results should verify if the main goals of AIDA2007 have been met. This means advancing the state-of-the-art in the following areas of abstract interpretation:

WP1: Abstract Interpretation Design

- disjunctive abstract domains for abstract model checking and abstraction refinements
- scalable abstract domains for pointer and numeric variable analyses
- complete abstract semantics and completeness-driven semantic transformations
- quantitative measures of the precision of abstract domains

WP2: Static Analysis by Abstract Interpretation

- analysis of causal properties of concurrent and distributed systems
- analysis of termination and complexity of C and Java programs
- analysis of string cleanness for C programs
- modular and generic analysis frameworks that handle annotation of programs
- speculative optimization through probabilistic abstract interpretation

WP3: Code Protection by Abstract Interpretation

- detection of obfuscated variants of malware's signatures
- code obfuscation of selected sensible properties
- software watermarks in loops and completeness holes

WP4: System Verification by Abstract Interpretation

- counterexample-guided abstraction refinement by abstract interpretation
- transformations of temporal languages in order to achieve strong preservation
- abstract interpretation-based algorithms for computing behavioural equivalences

The verification of these goals should be mostly based on research papers that will be presented at international peer-reviewed conferences and workshops or that will be published on international journals.

We also expect to develop new software tools and to improve existing ones like PPL, Julia and BinTerm. In particular, we expect to implement and evaluate the abstract domains and the static analysis techiques designed within WP1 and WP2. All these tools will be made available and, whenever possible, they will be released under the GNU GPL license.

One major purpose of AIDA2007 is to mobilize, unify and stimulate the Italian abstract interpretation community. In particular, our aim is to foster collaboration between members of different research units, especially within WP1 and WP2. The success of this collaboration may be measured in terms of joint papers and contributions, exchange of benchmarks, tools and software libraries, and organization of project meetings.

In order to ease the evaluation process, we will provide a web site for collecting all the research products of AIDA2007 (papers, tools, benchmarks, etc.).

17 - Mesi persona complessivi dedicati al Progetto di Ricerca

		Numero	Impegno 1° anno	Impegno 2° anno	Totale mesi persona
Componenti della sede dell'Unità di Ricerca		*7*	*55*	*51*	*106*
Componenti di altre Università /Enti vigilati		*1*	*8*	*6*	*14*
Titolari di assegni di ricerca		*2*	*16*	*14*	*30*
Titolari di borse	*Dottorato*	*1*	*10*	*10*	*20*
	Post-dottorato	*0*			
	Scuola di Specializzazione	*0*			

Personale a contratto	*Assegnisti*	*1*	*8*	*3*	*11*
	Borsisti	*1*	*3*	*9*	*12*
	Altre tipologie	*1*	*3*	*9*	*12*
Dottorati a carico del PRIN da destinare a questo specifico progetto		*0*	*0*	*0*	*0*
Altro personale		*0*			
TOTALE		**14**	**103**	**102**	**205**

18 - Costo complessivo del Progetto articolato per voci

Voce di spesa	Unità I	Unità II	Unità III	TOTALE
Materiale inventariabile	*4.500*	*8.000*	*4.500*	17.000
Grandi Attrezzature	*0*	*0*	*0*	0
Materiale di consumo e funzionamento	*4.500*	*2.000*	*4.500*	11.000
Spese per calcolo ed elaborazione dati	*0*	*0*	*0*	0
Personale a contratto	*20.000*	*19.000*	*20.000*	59.000
Dottorati a carico del PRIN da destinare a questo specifico progetto	*0*	*0*	*0*	0
Servizi esterni	*0*	*0*	*0*	0
Missioni	*13.500*	*23.000*	*13.500*	50.000
Pubblicazioni	*0*	*0*	*0*	0
Partecipazione / Organizzazione convegni	*2.500*	*8.000*	*2.500*	13.000
Altro	*0*	*0*	*0*	0
TOTALE	**45.000**	**60.000**	**45.000**	**150.000**

19 - Prospetto finanziario suddiviso per Unità di Ricerca

	Unità I	Unità II	Unità III	TOTALE
a.1) finanziamenti diretti, disponibili da parte di Università/Enti vigilati di appartenenza dei ricercatori dell'unità operativa	*2.300*	*18.000*	*0*	20.300
a.2) finanziamenti diretti acquisibili con certezza da parte di Università/Enti vigilati di appartenenza dei ricercatori dell'unità operativa	*11.200*	*0*	*13.500*	24.700
b.1) finanziamenti diretti disponibili messi a disposizione da parte di soggetti esterni	*0*	*0*	*0*	0
b.2) finanziamenti diretti acquisibili con certezza, messi a disposizione da parte di soggetti esterni	*0*	*0*	*0*	0
c) cofinanziamento richiesto al MUR	*31.500*	*42.000*	*31.500*	105.000
TOTALE	**45.000**	**60.000**	**45.000**	**150.000**

(per la copia da depositare presso l'Ateneo e per l'assenso alla diffusione via Internet delle informazioni riguardanti i programmi finanziati e la loro elaborazione necessaria alle valutazioni; D. Lgs, 196 del 30.6.2003 sulla "Tutela dei dati personali")

Firma _____ Data *(dal sistema alla chiusura della domanda)*