Honeywell
Honeywell Information Systems Italia

Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA
Via Viotti, 5 - Milano

23/24

NOTE DI SOFTWARE

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA

Honeywell
Honeywell Information Systems Italia

Note di software.
è un bollettino trimestrale, edito a cura del Centro di Ricerca e Progettazione della Honeywell Information Systems Italia e dell'Istituto di Cibernetica dell'Università di Milano.

Note di software.
intende costituire uno strumento per la rapida e informale circolazione di informazioni, idee ed esperienze nell'area della tecnologia del software. In particolare, intende stimolare il dibattito sulle metodologie orientate alla diminuzione del rapporto costo/qualità dei programmi, e sugli strumenti atti ad automatizzare l'attività di produzione del software in ogni suo aspetto.

La collaborazione a **Note di software** è libera.
In particolare, si sollecitano contributi nella forma di brevi monografie e bibliografie essenziali commentate sui seguenti temi: tecniche e strumenti per la programmazione strutturata e modulare, metodologie e strumenti per il testing e il debugging dei programmi, documentazione del software, aspetti organizzativi e gestionali nella costruzione di sistemi software di grandi dimensioni.

Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Comitato di Redazione (HISI - Via ai Laboratori Olivetti - Pregnana Milanese), o di inviare direttamente il dattiloscritto in triplice copia.
I contributi non dovrebbero superare le quattromila parole.

I lavori accettati per la pubblicazione non vengono revisionati da un comitato tecnico. Pertanto, ogni contributo pubblicato riflette le opinioni dei suoi autori, e non necessariamente quelle del Comitato di Redazione.
Eventuali revisioni saranno effettuate di comune accordo fra il Comitato di Redazione e i proponenti.

Note di software.
Viene distribuito a chi ne faccia richiesta alla redazione.

Direttore responsabile: Paolo Ghelfi
Comitato di Redazione: Roberto Polillo - Wanda Anselmetti - Stefania Bandini
Redazione: Franco Soresini

23/24

Giugno 1984

Indice:

Autorizzazione del Tribunale di Milano n. 87 del 23 febbraio 1977

NOTE DI SOFTWARE

QUESTO NUMERO...

Questo numero doppio di NOTE DI SOFTWARE è dedicato ai sistemi esperti, alla grafica, alla "Software Science".

Il primo articolo, "I sistemi esperti" costituisce una introduzione alla serie di articoli sullo stesso tema presente in questo numero.
In esso, Stefania Bandini delinea gli aspetti fondamentali che possono avvicinare alla problematica dei sistemi esperti: che cosa sono, da dove vengono, come funzionano e quali sono i settori applicativi in cui possono rivelarsi utili.

Segue il lavoro "Ambienti di sviluppo per sistemi esperti", in cui M. Borghesi, M. Gallanti e A. Stefanini illustrano le caratteristiche di alcuni sistemi specifici per lo sviluppo di sistemi esperti. Al di là dei linguaggi di programmazione di uso generale, vengono presentate le caratteristiche di due sistemi di sviluppo derivati da specifici sistemi esperti, EMYCIN ed EXPERT, e di due sistemi concepiti indipendentemente da una specifica applicazione, OPS-5 e SAGE.

Nel terzo lavoro, "TANGIE. A story generating program", Graziella Tonfoni descrive i concetti che stanno alla base di TANGIE, un programma per la generazione di testi narrativi basata su una serie di unità narrative di base. TANGIE può essere definito un programma "intelligente", perchè è in grado di produrre storie coerenti o di evidenziare incoerenze presenti in una data storia.

Infine, in "Note sull'uso di sistemi esperti per la diagnostica di guasti nella strumentazione elettronica", M. Benedetti presenta alcune prospettive per lo studio di nuove metodologie orientate alla applicazione dei sistemi esperti nel campo della ricerca di guasti in apparecchiature.

Seguono due lavori su argomenti di "computer graphics".
Nel primo, "Gli standard grafici", Daniele Marini presenta una sintetica rassegna sulla problematica degli standard grafici. Si illustrano brevemente le proposte CORE e GKS come standard per la realizzazione di pacchetti grafici applicativi, e dello standard IGES per l'interscambio di dati grafici tra differenti sistemi.

Nel secondo, "Un editor per Reti di Petri in una metodologia di costruzione di sistemi complessi", E. Ciapessoni e N. Negri descrivono un editor per Reti di Petri, basato sui principi della Interactive Computer Graphics, e su alcune particolari forme di interazione per rendere lo strumento estremamente flessibile.

Infine, chiude il numero "Software Science: una metrica per il software", di E. Bertolotti, un interessante lavoro di rassegna che presenta in modo sintetico i fondamenti teorici e alcuni tra i più significativi risultati sperimentali della "Software Science". Tale teoria, nata alcuni anni fa da un'idea di M. Halstead, si prefigge di stabilire le relazioni in grado di quantificare diverse ed importanti proprietà degli algoritmi, come la complessità, intesa come difficoltà di comprensione, il tasso di errori presenti in un programma, il tempo di sviluppo, e così via.

La redazione

I SISTEMI ESPERTI

Stefania Bandini
Istituto di Cibernetica – Università di Milano

RIASSUNTO

In questo lavoro vengono delineati sinteticamente gli aspetti fondamentali che possono avvicinare alla problematica dei Sistemi Esperti: che cosa sono, da dove vengono, come funzionano e quali sono i settori applicativi in cui possono rivelarsi utili. In appendice una tabella generale dei Sistemi Esperti più noti realizzati nel mondo, divisa per aree d'applicazione.

ABSTRACT

This paper shows shortly the fundamental aspects giving an overview on Expert Systems: what are they, where do they come from, how do they operate and what are the applicative areas where they can be useful instruments. In the appendix a general table about the more known Expert Systems realized in the world.

1. CHE COSA SONO

Questi ultimi anni hanno visto nascere ed evolversi un insieme di nuove tecniche di programmazione dei calcolatori, che ne ha esteso le potenzialità applicative in direzione di problemi considerati da sempre di stretta ed unica competenza umana.

Campi come la diagnostica (da quella medica a quella dei grandi impianti industriali), la prospezione geologica, l'ingegneria genetica o la configurazione di calcolatori (solo per citare qualche esempio tra i più rilevanti), caratterizzati da problemi che richiedono decisioni da parte di esperti umani, sono entrati a far parte della possibilità applicative dei calcolatori: i Sistemi Esperti si collocano proprio in questa area, dato che sono sistemi che possono raggiungere il livello di competenza di un esperto umano nella valutazione e nella soluzione di problemi specialistici.

Questi programmi differiscono dai comuni di programmi di calcolo in quanto sono progettati come assistenti "intelligenti" nella soluzione di problemi complessi.

Tradizionalmente i calcolatori sono utilizzati in quei settori in cui le capacità umane sono limitate: i compiti che devono svolgere, essenzialmente, riguardano l'effettuazione di calcoli complessi e la memorizzazione di grandi quantità di dati. I relativi programmi realizzano algoritmi: procedure completamente definite, passo passo, per risolvere specifici problemi. Tuttavia, i programmi deterministici risolvono solo una frazione dei problemi umani; determinate applicazioni richiedono, invece, la progettazione di macchine in grado di effettuare deduzioni o, quanto meno, di riprodurre alcuni modi in cui l'uomo utilizza la sua passata esperienza e le sue conoscenze per risolvere nuovi problemi.

La specifica caratteristica dei Sistemi Esperti consiste appunto nella loro capacità di trarre conclusioni da un insieme di conoscenze relative ad una ben precisa classe di problemi, registrate in una memoria sotto forma di regole simili a quelle usate dagli esperti umani per prendere decisioni nel loro campo di competenza.

La conoscenza è rappresentata internamente in forma simbolica, tramite un linguaggio di rappresentazione appositamente definito e il Sistema Esperto la interpreta e la manipola per trarre conclusioni utilizzando un meccanismo di deduzione basato, su principi della logica o su loro estensioni.

Dal fatto che la conoscenza ha una rappresentazione in forma esplicita (simbolica) anzichè codificata in procedure compilate, consegue che il Sistema Esperto è in grado, su richiesta, di spiegare, il proprio processo di ragionamento e di rendere conto all'utente delle conoscenze e dei meccanismi deduttivi utilizzati. Questa qualità è detta "trasparenza", in opposizione alla "opacità" di un normale programma che generalmente nasconde all'utente il proprio modo di funzionamento.

Il meccanismo deduttivo di un Sistema Esperto non segue un albero di decisione predefinito: esso è tipicamente non deterministico poichè può esplorare diverse alternative nel processo di ricerca di una conclusione; il sistema soppesa ad ogni passo di funzionamento fatti ed ipotesi e, in un certo senso, effettua la scelta più appropriata nel contesto specifico del problema che gli è presentato.

2. DA DOVE VENGONO

I Sistemi Esperti sono uno dei prodotti più convincenti dal punto di vista applicativo dell'Intelligenza Artificiale, una disciplina nata nella metà degli anni '50 nei laboratori di ricerca universitaria degli Stati Uniti con l'obiettivo di "far fare al calcolatore attività che richiederebbe intelligenza se fatte dall'uomo", secondo le parole di uno dei fondatori della disciplina, Marvin Minsky del Massachusetts Institute of Technology (MIT).

Le metodologie e le tecniche dell'Intelligenza Artificiale furono applicate inizialmente a problemi esemplari, ma di limitato interesse applicativo (giochi, risoluzione di problemi astratti, sistemi di dimostrazione di teoremi); col tempo, lo spettro dei campi applicativi si è allargato fino a divenire un fattore primario nello sviluppo delle applicazioni più avanzate dell'informatica (robotica, comprensione del linguaggio naturale, elaborazione della voce, Sistemi Esperti, sintesi di programmi, visione, etc.).

Uno degli aspetti che è di importanza centrale in quasi tutte queste applicazioni è il problema di come rappresentare in forma simbolica la conoscenza del campo e come sviluppare su di essa attività di ragionamento: il tema della rappresentazione della conoscenza riguarda aspetti sia teorici che implementativi. Nell'ambito dell'Intelligenza Artificiale sono state sviluppate teorie, metodologie e strumenti per trattare questo problema e tale corpo di nozioni ha fornito a questa disciplina gran parte del suo status concettuale.

Inizialmente, i riceratori dell'Intelligenza Artificiale si posero obiettivi estremamente ambiziosi, come la costruzione di un "Risolutore Generale di Problemi". Questi studi non portarono a risultati convincenti dal punto di vista applicativo (se non relativamente a problemi di natura teorica). Consentirono, tuttavia, di mettere a fuoco il fatto che era possibile risolvere con sistemi automatici problemi di una certa complessità solo restringendo opportunamente il campo applicativo (problemi specifici di determinati campi della conoscenza umana), ad esempio, impiegando metodi rivolti alla codificazione di una conscenza organizzata di un ben delimitato settore e, parallelamente, costruendo meccanismi di ragionamento di tipo generale ma parametrizzabili al fine di ottenere risultati significativi nel contesto.

Si giunse così alla realizzazione dei primi prototipi di Sistemi Esperti, nati dalla feconda collaborazione tra riceratori dell'Intelligenza Artificiale e ricercatori nel campo della medicina e della chimica, in grado di fornire problemi specifici e ben delimitati rispetto alla loro portata e la conoscenza esperta indispensabile per risolverli.

Vogliamo ricordare DENDRAL come esempio molto significativo, non solo perchè storicamente deve essere considerato come il primo Sistema Esperto, ma anche perchè il suo successo applicativo segnò una tappa fondamentale, determinando gli indirizzi di tutto il settore.

DENDRAL è stato progettato e realizzato da Edward Feigenbaum in collaborazione con il Nobel della chimica Lederberg verso la fine degli anni '60: è un Sistema Esperto che ha il compito di identificare la formula strutturale di un composto chimico, partendo da dati di spettrometria di massa e da informazioni fornite dal consultante.

Per molecole di una certa complessità, esistono milioni e milioni di possibili formule strutturali corrispondenti ad una certa formula bruta; il chimico seleziona determinate possibili strutture in base ad ipotesi, frutto della sua esperienza nel campo, in accordo con informazioni a carattere qualitativo deducibili dallo spettro.

La competenza di un chimico consiste in una serie di regole costituenti una "euristica", cioè una metodologia volta a trovare (in greco, "eurisko") soluzioni a problemi specifici: la conoscenza euristica consiste nella pratica continua in un settore e comprende metodologie risolutive rivolte alla ricerca di soluzioni per tentativi, e valutazioni qualitative che possono avere un carattere approssimativo.

Tale metodologia viene espressa nella formalizzazione della conoscenza tramite *regole di produzione,* di cui il seguente può essere un caso esemplificativo:

"Se lo spettro ha due picchi, alle masse X1 e X2, tali che

a. X1 + X2 = M + 28
b. X1 − 28 è un picco alto
c. X2 − 28 è un picco alto
d. almeno uno tra X1 e X2 è alto

allora la molecola contiene un gruppo chetone"

Questo tipo di regole di carattere euristico sono utilizzate da DENDRAL sia per indirizzare il procedimento di generazione delle strutture candidate che per stabilirne il rango di credibilità in accordo con i dati a disposizione.

DENDRAL è attualmente impiegato in laboratori di chimica e vengono ad esso attribuite prestazioni di altissima qualità.

3. COME FUNZIONANO

È difficile ricondurre gli ormai numerosi esempi di Sistemi Esperti ad una struttura unitaria. È tuttavia possibile riconoscere nella larga maggioranza di tali sistemi, le seguenti parti funzionali (fig. 1):

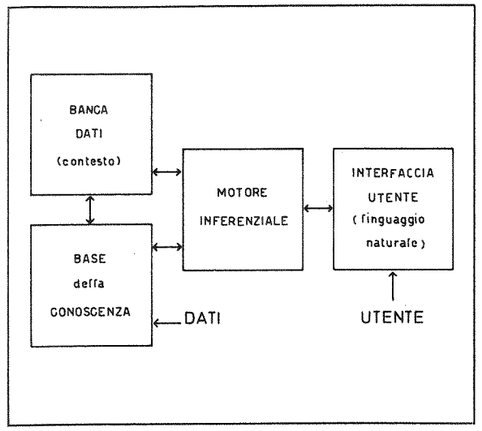


Fig. 1

- una memoria di lavoro, contenente la rappresentazione corrente del problema oggetto della consultazione, in sintesi i fatti noti e le ipotesi correnti indagate;

- una base della conoscenza, contenente un insieme di relazioni, codificate in forma simbolica, tra fatti e ipotesi o fra ipotesi e ipotesi;

- un meccanismo deduttivo o di inferenza, che ha il compito di confrontare la corrente rappresentazione del problema con le relazioni costituenti la base della conoscenza, al fine di identificare situazioni, selezionare ipotesi e, in definitiva, costruire le soluzioni al problema affrontato;

- una interfaccia uomo-macchina, che ha la duplice funzione di assistere l'utente durante la consultazione e di consentire la introduzione, la verifica di consistenza, il debugging e la modifica della base della conoscenza da parte dell'esperto.

Il modulo più critico di un Sistema Esperto è la sua base di conoscenza, dalla quale dipende la qualità delle prestazioni del sistema.

Caratteristica specifica di buona parte della conoscenza di un esperto è la sua natura empirica, non formalizzata. È evidente che la sua traduzione in base della conoscenza comporta un processo di analisi e di formalizzazione, che al momento viene effettuata da parte dell'analista, meglio conosciuto come Ingegnere della Conoscenza, attraverso una serie di colloqui con l'esperto del settore (fig. 2).

L'ingegnere della conoscenza si incarica quindi di individuare la struttura del problema da risolvere, di idendificare e tradurre in linguaggio simbolico le associazioni e i processi deduttivi dell'esperto.

Tra le tecniche di rappresentazione della conoscenza nate negli ambienti dell'Intelligenza Artificiale, la più utilizzata nei Sistemi Esperti è quella che fa riferimento alle *regole di produzione.*

Si tratta di regole dichiarative del tipo "premessa ⇒ conclusione" (SE è verificato A, ALLORA se ne conclude B), che insieme compongono la base della conoscenza del Sistema Esperto.

Il motore inferenziale è un meccanismo di ricerca e selezione che agisce su questa base; il suo funzionamento è ciclico, in quanto si basa sù operazioni del tipo "riconoscimento - attivazione": dopo un confronto tra i dati correnti del problema con la premessa (parte "SE" dalla regola di produzione), giunge alla selezione di un certo numero di regole pertinenti.
Tra queste, il motore inferenziale ha il compio di individuare la regola più appropriata e di passare quindi all'attivazione della conclusione (parte "ALLORA" della regola di produzione), modificando conseguentemente lo stato corrente del problema in esame.

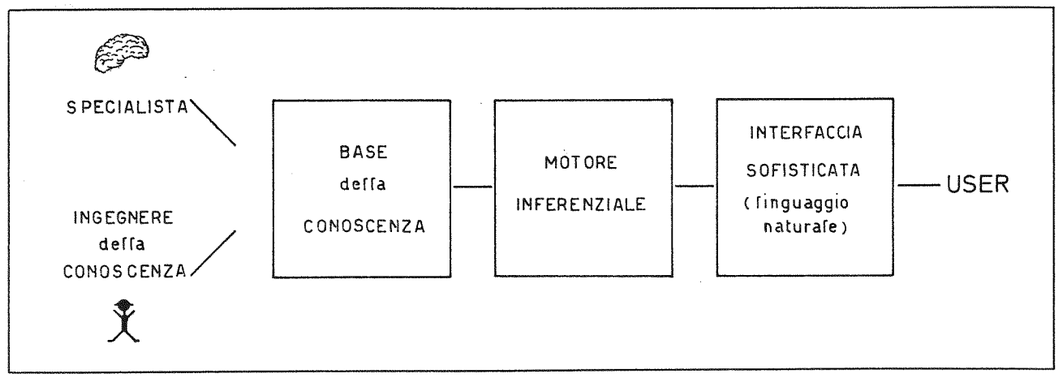


Fig. 2

Gli algoritmi di ricerca e selezione che agiscono sulla base della conoscenza, come si può facilmente intuire, possono raggiungere una notevole complessità. In linea di massima, comunque, tali algoritmi si conformano a due strategie abbastanza lineari:

- *ricerca in avanti* (forward): la selezione delle regole da utilizzare avviene sulla base dei dati di partenza accertati. Vengono scelte le regole che contengono nelle premesse riferimenti a quei fatti il cui valore è noto al ciclo n-esimo.

- *ricerca all'indietro* (backward): le regole da utilizzare vengono selezionate sulla base dell'obiettivo che si vuole raggiungere al ciclo n-esimo (generalmente un'ipotesi da verificare). Vengono scelte le regole che contengono nella conclusione l'obiettivo desiderato: se la loro premessa è vera, l'obiettivo è raggiunto, altrimenti tale premessa viene assunta come la nuova ipotesi da provare al ciclo successivo (n+1-esimo).

Queste strategie, in apparenza di esecuzione lineare, vengono complicate da fattori di carattere probabilistico. In generale, infatti, le regole di produzione, per rappresentare il carattere empirico di certe conoscenze, collegano la conclusione alla premessa con un fattore di probabilità variabile in un certo intervallo i cui estremi rappresentano il dubbio totale e la assoluta certezza.

Questo significa che, ad ogni attivazione di una regola, la plausibilità di un ipotesi o di una conclusione varia in base al fattore di certezza associato alla regola.

Un altro fattore non trascurabile per la gestione di strategie di selezione delle regole di produzione è quello che riguarda l'uso di *metaregole,* anch'esse di natura euristica, che forniscono al meccanismo di inferenza conoscenze aggiuntive, utili a meglio dirigere la scelta dei passi da compiere.

4. I CAMPI DI APPLICAZIONE

Come abbiamo visto, la base della conoscenza di un Sistema Esperto è costituita da un insieme di regole di produzione, che stabiliscono relazioni tra determinati fatti e determinate ipotesi.

Non esistono a priori legami tra una regola di produzione e un'altra: questa caratteristica consente di rappresentare efficacemente domini in cui la conoscenza è effettivamente costituita da un insieme di relazioni sparse, invece che da una teoria unitaria che inquadra e spiega ogni singolo fenomeno; si tratta, specificatamente, del tipo di situazione che caratterizza quasi tutti i settori in cui viene richiesta una specifica conoscenza esperta.

L'introduzione dei fattori di certezza risponde alla esigenza di rappresentare relazioni a carattere empirico e consente l'introduzione di regole più specifiche che contemplino le eccezioni accanto a regole più generali.

La possibilità di contemplare eccezioni ("non monotonicità") caratterizza il tipo di rappresentazione della conoscenza proprio dei Sistemi Esperti in opposizione alle rappresentazioni basate sull'ordinaria logica del primo ordine.

L'assenza di dipendenze funzionali a priori tra le regole di produzione consente inoltre di modificare in modo relativamente poco costoso una base di conoscenze, sostituendo o aggiungendo regole e quindi determinando un accrescimento incrementale della base di conoscenze e, conseguentemente, delle prestazioni del sistema.

Questa caratteristica rende i Sistemi Esperti di grande interesse non soltanto in settori nei quali la conoscenza specifica ha effettivamente il carattere di un insieme di relazioni di natura empirica, come in molte delle applicazioni che abbiamo considerato (medicina, prospezione mineraria, ricerca guasti, e in genere settori in cui il processo decisionale ha carattere di diagnosi), ma anche laddove il processo decisionale è un albero così complesso che la sua rappresentazione mediante un algoritmo si traduce in un programma inefficiente e poco flessibile.

5. SETTORI PER SISTEMI ESPERTI

Vediamo ora una rassegna delle principali classi di Sistemi Esperti dal punto di vista dei problemi che devono affrontare e risolvere.

- **Interpretazione**

I Sistemi Esperti che hanno funzioni interpretative sono volti alla ricerca di un significato da attribuire ad un insieme di dati.

Le difficoltà che investono le operazioni di interpretazione nascono dal fatto che spesso il dati sono accompagnati da errori o da rumore e perciò l'interpretazione può essere effettuata solo su informazioni parziali o, spesso, anche contradditorie tra loro.

Un sistema di interpretazione, poi, può incontrare situazioni in cui sono presenti più significati attribuibili ad un insieme di dati e risulta perciò necessaria una meticolosa analisi di ognuno di essi per poter scartare meno significativi.

I settori in cui un sistema di intepretazione può rivelarsi un valido strumento vanno dai problemi della sorveglianza a quelli della comprensione del parlato

e dell'analisi di immagini o di strutture chimiche, fino alla interpretazione di segnali di controllo.

- **Diagnostica**

La diagnostica è uno dei terreni che più ha offerto spunti alla progettazione e alla realizzazione di Sistemi Esperti; dal campo medico a quello dei grandi impianti industriali, i problemi di malfunzionamento accomunano le metodologie di ricerca e di diagnosi e di terapie opportune sia per i sistemi viventi che per quelli artificiali.

In campo medico, infatti, le diagnosi vengono raggiunte con l'interpretazione di dati (sintomi) in base alla conoscenza del sistema funzionante (anatomia, fisiologia, etc.)

Questo tipo di stilizzazione di un metodologia, come appare evidente, può riguardare anche il mondo dei macchinari e delle apparecchiature industriali a cui è infatti rivolta molta dell'attenzione dei progettisti di Sistemi Esperti: dal caso di rotori per la produzione di energia elettrica, all'analisi dei motori diesel per locomotive o navi fino ai sistemi di shut-down delle piattaforme per la ricerca del petrolio.

- **Previsione**

I Sistemi Esperti usati per la previsione forniscono un valido supporto a fronte di problemi in cui bisogna inferire conseguenze da situazioni date.

Una delle maggiori difficoltà che si incontrano in questi casi riguarda la quantità, la affezione da rumore e la frequente incompletezza dei dati su cui operare previsioni: le previsioni demografiche o socio-economiche, come le previsioni sul traffico (aereo, marittimo, etc.) sono degli esempi illuminanti per quel che concerne la natura dei problemi da affrontare.

- **Progettazione**

I Sistemi Esperti impiegati nella progettazione hanno il compito di configurare oggetti in modo da soddisfare le condizioni poste dal problema.

Possiamo includere in questa categoria, per esempio, la realizzazione di circuiti o di progetti di costruzioni.

Una delle esigenze che questi sistemi devono soddisfare riguarda la valutazione dei costi o di altre indesiderabili proprietà dell'oggetto disegnato da parte dell'utente.

- **Monitoraggio**

I Sistemi Esperti impiegati nell'interpretazione di dati provenienti da monitoraggio devono affrontare tutte le difficoltà già contemplate per quelli impiegati nella semplice interpretazione di dati e, in più, devono assolvere a tutti i compiti che derivano dall'uso dei dispositivi di rilevamento.

I sistemi di monitoraggio possono riguardare sia il controllo costante che l'ispezione effettuata in caso di necessità.

I settori d'applicazione di questo tipo di sistemi toccano un amplissimo spettro di casi: dagli impianti delle centrali nucleari, alle terapie intensive applicate ad ammalati (per es. polmone d'acciaio), alla gestione del traffico aereo, fino ai compiti di management fiscale.

Le caratteristiche comuni a queste applicazioni possono essere sintetizzate nei seguenti punti:

- i problemi presentano vasti spazi di risoluzione;

- spesso vi è una basilare esigenza di procedere per tentativi;

- non è raro che vengano trattati valori che subiscano variazioni nel tempo;

- spesso le informazioni e i dati sono accompagnati da alte percentuali di disturbo.

6. GIAPPONE E QUINTA GENERAZIONE

Nel 1981 il MITI (Ministero dell'Industria e del Commercio Internazionale) giapponese ha patrocinato la ormai famosa "Conferenza Internazionale sui Calcolatori della Quinta Generazione" in cui si sono gettate le basi per i progetti di ricerca per lo sviluppo di sistemi di calcolo per gli anni Novanta.

Il MITI ha infatti proposto un ambizioso piano di ricerca e sviluppo (con richiesta al governo di altissimi investimenti) per il "Progetto Quinta Generazione".

Il settore dei Sistemi Esperti, dopo i successi rilevati in questi ultimi anni, ha suscitato ampi interessi presso gli studiosi giapponesi, che già stanno lavorando in modo intenso e proficuo in questa direzione oltre che in tutte le aree dell'Intelligenza Artificiale.

La meta che si sono prefissati i componenti del "Progetto Quinta Generazione" è indirizzata verso la progettazione e la realizzazione di una macchina di alta potenza che ha caratteristiche intelligenti, in grado, cioè di compiere azioni considerate da sempre qualità esclusive dell'uomo.

Tale indirizzo prevede una combinazione tra ricerche VLSI, processi paralleli, pattern recognition, programmazione orientata al calcolo logico e sistemi basati sulla conoscenza.

7. UN ELENCO DEI PRINCIPALI SISTEMI ESPERTI

MEDICINA:

MYCIN — È stato realizzato per l'identificazione delle infezioni batteriche del sangue e per l'assistenza nelle decisioni terapeutiche da prendere per combatterle.
(E.H. Shortliffe - Stanford University)

TERESIAS — Più che di un Sistema Esperto vero e proprio è un supporto a MYCIN che permette l'acquisizione automatica di nuove regole mediante dialogo con l'utente.
(R. Davis, D. Lenat - Stanford University)

INTERNIST — È un sistema per la consultazione nel campo della medicina interna.
(J.D. Myers, H.E. Pople - University of Pittsburg)

PUFF — Si tratta di un sistema che fornisce l'interpretazione di dati per il controllo della funzionalità polmonare.
(J. Kunt - Stanford University)

CASNET — È un sistema di diagnosi rivolto al campo dell'oculistica che assiste nell'identificazione di glaucomi.
(S. Weiss, C. Kulikowski - Rutgers University)

VM — Questo sistema permette il monitoraggio in tempo reale di un paziente sottoposto a terapia intensiva (per es. polmone d'acciaio).
(L.M. Fagan - Stanford University)

LITO-1 — È un Sistema Esperto per la valutazione delle funzionalità epatiche. È stato sviluppato presso l'Istituto di Scienze dell'Informazione dell'Università di Torino.
(P. Torasso - Università di Torino)

CENTAUR — Interpreta misure ottenute da testing sul funzionamento polmonare di pazienti affetti da disturbi al sistema respiratorio.
(J.S. Aikins - Stanford University)

HEADMED — È un sistema che viene utilizzato nei settori della psicofarmacologia.
(J.F. Heiser)

MDX — Si tratta di un sistema che assiste nelle diagnosi relative alla colostasi (ostacolo al deflusso della bile).
(B. Chandrasekaran - Ohio State University)

ONCONCIN — Assiste nel trattamento di protocolli chemioterapici nelle forme di linfoma in pazienti affetti da cancro.
(E. Shortliffe, A.C. Scott - Stanford University)

RX — Tratta conoscenze circa il decorso e il trattamento di malattie croniche utilizzando un database di informazioni sul paziente.
(R.L.Blum - Stanford University)

CHIMICA:

DENDRAL — Fornisce assistenza nell'analisi di spettrometrie di massa molecolare di composti organici.
(E.A. Feigenbaum, B.G. Buchnan - Stanford University)

CRYSALIS — Permette l'identificazione della struttura delle proteine utilizzando i dati provenienti da cristallografie.
(R. Engelmore, H.Nil - Stanford University)

SECS — Viene utilizzato nelle fasi di progettazione di sintesi organiche.
(W. Wipke)

CONGEN — È stato realizzato per la ricerca della struttura molecolare di composti sconosciuti.
(R.E. Chartart - Stanford University)

GEOLOGIA:

PROSPECTOR — Assiste nell'analisi su dati ricavati da prospezioni minerarie.
(P.E. Hart, R.O. Duda - SRI International)

DIPMETER ADVISOR — È utilizzato nella ricerca di giacimenti petroliferi.
(R. Davis - Massachusetts Institute of Technology - Schlumberger - Doll Research).

DITATTICA:

SOPHIE — È un sistema che guida lo studente nell'apprendimento di metodologie per la progettazione di circuiti elettronici.
(D. Sleeman).

GUIDON — Utilizza la base di conoscenza di MYCIN (malattie infettive del sangue), ma è orientato, però, all'insegnamento delle metodologie di diagnosi piuttosto che alla diagnosi in se stessa.
(W.J. Clancey - Stanford University)

INGEGNERIA:

SACON — Fornisce consulenze agli ingegneri sull'uso di procedure per l'analisi di strutture.
(J. Bennet, R. Engelmore - Stanford University)

EL — È un sistema che permette la sintesi di circuiti elettrici.
(G.J. Sussman - Massachusetts Institute of Technology)

LOGIN — Permette lo sviluppo di strumenti di complemento o di supplemento per segnalazioni nel campo di processazione di programmi.

SU/X — Formula ipotesi circa la locazione, la velocità, etc. di oggetti da dati provenienti da segnali.
(E.A. Feigenbaum, H.P. Nil - Stanford University)

MECCANICA RAZIONALE:

MECHO — È un sistema che risolve problemi di meccanica razionale.
(A. Bundy et al. - Edimburgh University)

DIAGNOSTICA INDUSTRIALE:

DART — Nato da una collaborazione tra IBM e l'Università di Stanford, fornisce tecniche avanzate per la diagnosi di guasti in sistemi di calcolo: è applicato al sistema IBM 43/41.
(J. Bennet, C. Hollander - Stanford University - IBM Corporation)

IDT — Si tratta di uno studio sulle metodologie da applicarsi nella diagnosi dei sistemi elettronici. È stato sviluppato in ambito Digital.
(H. Shubin et al. - Digital Equipment Corporation)

DELTA-CATS — Sistema sviluppato presso la General Electric che si occupa della ricerca di guasti e della riparazione di grandi motori diesel.
(P. Bonissone - General Electric)

SPERIL — Sistema per la valutazione di danni a strutture statiche che sono state soggette a sollecitazioni (es. terremoti); utilizza informazioni rilevate da ispezioni visive effettuate in vari punti della struttura.
(M. Ishizuka, K.Fu, J. Yao - Universiy of Tokyo)

AL/X — Diagnostica le cause di interruzioni da allarme su piattaforme per la ricerca petrolifera.
(J. Reiter - Intelligent Terminals Ltd. - University of Edinburgh)

CRIB — È utilizzato nella diagnostica di errori nello hardware e nel software dei calcolatori.
(T.R. Addis - International Computers Limited)

RAFFLES — Anche questo sistema rigurada la diagnosi nello hardware e nel software dei calcolatori.
(T.R. Addis - International Computers Limited)

CONFIGURAZIONE DI CALCOLATORI:

R1 — È utilizzato per configurare il calcolatore VAX 11/780 partendo dai moduli di base che lo compongono. Questo sistema è stato sviluppato presso la Carnegie-Mellon University per conto della Digital Equipment Corporation.
(J. Mc.Dermott - Carnegie-Mellon University - Digital Equipment Corporation)

XSEL — Realizza un front-end d R1; coadiuva il venditore per fornire una configurazione hardware/software per il VAX 11/780 sulla base delle esigenze del cliente.
(J. Mc.Dermott - Carnegie Mellon University)

MATEMATICA:

AM — È un sistema costruito per la scoperta automatica di teoremi.
(D.B. Lenat - Stanford University)

MACSIMA — Viene utilizzato nella manipolazione simbolica per i settori matematici (algebra).
(M.R. Genesereth - Massachusetts Institute of Technology)

SCIENZA:

BACON 4 — È stato relizzato per la scoperta di leggi scientifiche di carattere empirico.
(P. Langley, G. Bradshaw - Carnegie Mellon University)

CONCHE — Si tratta di un sistema che coadiuva nella formazione di teorie scientifiche.
(I.H. Chisholm, D.H. Sleeman - University of Leeds)

GAMMA — Fornisce l'interpretazione di spettri ottenuti mediante attivazioni di raggi gamma.
(D.R. Barstow - Yale University)

MOLGEN — Mette a disposizione strumenti per lo studio della genetica molecolare nella pianificazione di sperimentazioni su manipolazioni del DNA.
(J. Lenderberg, M. Stefik et al. - Stanford University)

8. BIBLIOGRAFIA

[1] A. Barr, E.A. Feigenbaum (Eds), THE HANDBOOK OF ARTIFICIAL INTELLIGENCE, 3 voll., William Kaufmann Inc., Los Altos, CA, 1981

[2] N.J. Nillson, PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Tioga, Palo Alto, CA, 1980

[3] P.H. Winston, ARTIFICIAL INTELLIGENCE, Addison-Wesley, Reading, MA 1977

[4] P.H. Winston, R.H. Brown (Eds), ARTIFICIAL INTELLIGENCE. AN MIT PERSPECTIVE, MIT Presds, Cambridge, MA, 1979

[5] D. Mitchie (Ed), EXPERT SYSTEMS IN THE MICRO ELECTRONIC AGE, Edinburgh University Press, 1979

[6] M. Stefik et al., THE ORGANIZATION OF EXPERT SYSTEMS: A PRESCRIPTIVE TUTORIAL, Xerox, Palo Alto, CA, 1982.

[7] B.G. Butchnan, NEW RESEARCH ON EXPERT SYSTEMS, "Artificial Intelligence Magazine", n. 10, 1982

[8] AA.VV. A NEW TECHNICAL STYDY: ARTIFICIAL INTELLIGENCE - A NEW TOOL FOR INDUSTRY AND BUSINESS, Technical Insight Inc., Fort Lee, NY, (pagg. 159, $520)

[9] R. Davis et al., PRODUCTION RULES AS A REPRESENTATION OF A KNOWLEDGE-BASED CONSULTATION PROGRAM, , "Artificial Intelligence", vol. 8, n.1, 1977

[10] R. Davis, J. King, AN OVERVIEW OF PRODUCTION SYSTEM, in "Machine representation of knowledge", D. Reidel Publ. Co., Dordrecht, 1976

[11] K. Fuchi, THE DIRECTION THE FGCS PROJECT WILL TAKE, "New Generation Computing", n. 1, Springer-Verlag Ed., Berlino, New York, 1983

[12] D.S. Nau, EXPERT COMPUTER SYSTEMS, "Computer", Feb. 1983

[13] G. Guida, C. Tasso, IL GRANDE FRATELLO?, "Sistemi e atomazione", n. 230, Ott. 1982

[14] F. Gardin, INTELLIGENZA ARTIFICIALE E SISTEMI ESPERTI, "BIT", n. 32, n. 34, 1982, n. 36, 1983

[15] S. Bandini, M. Gallanti, BIBLIOGRAFIE: GLI EXPERT SYSTEMS, "Note di software", n. 21/22, dicembre, 1983

[16] E.A. Feigenbaum, P. Mc Corduck, THE FIFTH GENERATION, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983

[17] F. Hayes-Roth, D.A. Waterman, D.B. Lenat, BUILDING EXPERT SYSTEMS, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983

AMBIENTI DI SVILUPPO PER SISTEMI ESPERTI

M. Borghesi, M. Gallanti, A. Stefanini
CISE - Segrate

RIASSUNTO

L'articolo illustra le caratteristiche di alcuni strumenti specifici per lo sviluppo di sistemi esperti. Al di là dei linguaggi di programmazione di uso generale, vengono presentate le caratteristiche di due sistemi di sviluppo derivati da specifici sistemi esperti, EMYCIN ed EXPERT, e di due sistemi concepiti indipendentemente da una specifica applicazione, OPS-5 e SAGE.

ABSTRACT

This paper presents the main characteristics of some Expert System development tools.
Beyond general purpose programming languages, we examine two development tools deriving from previous Expert Systems: EMYCIN and EXPERT, and two systems that were not designed for a specific application domain: OPS-5 and SAGE.

1. INTRODUZIONE

Nella realizzazione di un sistema esperto il compito più rilevante ed impegnativo consiste nella costruzione della base di conoscenza. Allo stato attuale della tecnologia del settore, questo compito è affidato ad una figura nota come *ingegnere della conoscenza,* e consiste essenzialmente nel formalizzare in un apposito linguaggio la conoscenza dell'esperto.

L'ingegnere della conoscenza procede per affinamenti successivi, effettuando una analisi iniziale che porta a sviluppare un primo modello, ancora grossolano e non formale, del dominio di conoscenza, e quindi procedendo alla realizzazione di un primo nucleo della base di conoscenza, che può corrispondere ad un sottoinsieme del problema, e infine ampliando questo nucleo iniziale sia in estensione che in profondità, sino ad ottenere un sistema la cui affidabilità di giudizio sia paragonabile a quella dell'esperto umano.

L'implementazione della base di conoscenza presuppone l'identificazione di un linguaggio formale di rappresentazione, che consenta di esprimere proprietà e relazioni fra gli oggetti del dominio.

Il lavoro è stato svolto nell'ambito di una commessa di ricerca finanziata dall'ENEL.

Questa esigenza si intreccia fortemente con la scelta dello strumento di sviluppo che si intende adottare. Si possono infatti seguire due approcci divergenti:

1. sviluppare liberamente un proprio formalismo di rappresentazione. La costruzione del sistema esperto comporta dunque anche la realizzazione dell'interprete del linguaggio di rappresentazione, e postula l'impiego di un linguaggio di programmazione per uso generale. I vari dialetti del Lisp sono candidati ideali per questo compito, come per altre applicazioni di Intelligenza Artificiale, per la singolare combinazione di strumenti di basso livello, libertà espressiva, e potenti primitive per la manipolazione di liste di simboli che il linguaggio mette a disposizione.

2. appoggiarsi al linguaggio di rappresentazione adottato in un sistema esperto esistente. Di molti sistemi esperti è infatti disponibile una versione priva della specifica base di conoscenza. Tale versione può essere utilizzata quale base per lo sviluppo di altri sistemi.
   Questo tipo di strumenti è variamente denominato (conchiglia, guscio, sistema scheletro): nel presente lavoro sarà indicato come *sistema di sviluppo.*

In questo lavoro vengono presentate le caratteristiche di alcuni sistemi di sviluppo fra i più noti e disponibili.
EMYCIN [1] è il primo e più noto esempio di sistema di sviluppo.'Esso deriva da MYCIN [2], un sistema esperto ormai classico, sviluppato nel corso degli anni '70 all'Università di Stanford. EMYCIN – al quale sarà dedicata una sezione del presente lavoro – è stato utilizzato come base per numerosi altri sistemi esperti (fra i più noti citiamo PUFF [3], per la diagnosi di malattie polmonari, e SACON [4], un'assistente all'impiego di packages per il calcolo di strutture.

Alcuni altri sistemi esperti hanno generato sistemi di sviluppo.Fra i più noti citiamo PROSPECTOR [5] da cui è derivato KAS, e CASNET [6] da cui è stato tratto EXPERT [7], che viene anch'esso trattato in dettaglio nel seguito.

Un sistema di sviluppo mette a disposizione l'interprete di un linguaggio con il quale si esprimono essenzialmente:

1. gli oggetti del dominio di conoscenza, e più specificatamente quegli attributi di tali oggetti che sono rilevanti ai fini del problema oggetto della consultazione;

2. le regole che correlano ad una premessa, consistente in una determinata combinazione di valori dei predetti attributi, una specifica azione da compiere su uno o più oggetti.

Nel corso di una consultazione, i valori di alcuni attributi vengono assegnati sulla base di domande poste all'utente, mentre altri, che costituiscono l'obiettivo dell'indagine, vengono dedotti mediante le regole che formano la base di conoscenza.

Il sistema di sviluppo fornisce quindi anche un supporto alla consultazione, la cui parte essenziale è il meccanismo inferenziale, cioè l'algoritmo che realizza una determinata strategia per la ricerca degli obiettivi tramite l'individuazione delle regole pertinenti e la selezione delle regole applicabili in un determinato contesto.

Evidentemente il formalismo di rappresentazione ed il meccanismo inferenziale messi a disposizione da un sistema di sviluppo risultano modellati sulle caratteristiche del dominio originario per cui il sistema esperto era stato concepito. L'adozione di un sistema di sviluppo può rendere quindi molto spedito il compito dell'ingegnere della conoscenza, ma può anche presentare limiti consistenti, talora insuperabili, che, per la natura del processo di ingegneria della conoscenza, che procede per accrescimenti e raffinamenti successivi, possono anche non risultare evidenti sin dall'inizio.

Accanto ai sistemi di sviluppo derivati da sistemi esperti specifici, quali quelli citati in precedenza, ve ne sono altri che non sono nati da un sistema preesistente. Nel corso di questa rassegna esamineremo in dettaglio, le caratteristiche di SAGE [9], uno strumento sviluppato da SPL International (1) indipendentemente da applicazioni specifiche, e di OPS5 [10], realizzato presso la Carnegie Mellon University.

OPS5 è stato utilizzato per lo sviluppare R1 [12], un sistema esperto per la configurazione di calcolatori DEC VAX, che ha avuto una larga risonanza ed è attualmente impiegato sulle linee di produzione Digital. La genesi di OPS5 è, tuttavia, indipendente da R1: esso è uno dei membri della famiglia di linguaggi OPS, tuttora in evoluzione.

(1) - SPL è un'importante società di servizi software inglese, attiva nel campo del software di sistema e per il controllo di processo, nell'ambito della quale opera da anni un consistente gruppo di ingegneria della conoscenza.

2. EMYCIN

EMYCIN [1] è un derivato di MYCIN [2], uno dei più noti sistemi esperti, progettato da E. Shortliffe e altri dell'Università di Stanford, per la diagnosi e la terapia delle infezioni del sangue.

Privando MYCIN della conoscenza relativa allo specifico dominio di applicazione originario, si è ottenuto uno strumento particolarmente adatto a problemi soggetti ad una indagine deduttiva come la diagnosi di guasto, nella quale è di solito disponibile un largo spettro di dati di ingresso, talora poco affidabili (sintomi, prove di laboratorio...), e le possibili soluzioni possono essere enumerate.

EMYCIN consente di applicare il meccanismo deduttivo di MYCIN a nuovi problemi in cui la conoscenza specifica può essere rappresentata con il linguaggio per esprimere le regole proprio di MYCIN, mettendo a disposizione tutti gli strumenti di supporto di MYCIN (come ad esempio la sua versatile interfaccia per la spiegazione), che facilitano grandemente la costruzione e la verifica di una base di conoscenza.

Un sistema derivato da EMYCIN è stato recentemente posto in commercio da Teknowledge, la più importante società americana per lo sviluppo di sistemi esperti, sotto il nome di KS-300.
KS-300 è disponibile in ambiente UNIX di Berkeley, su VAX e su altre macchine.

2.1.1 La rappresentazione della conoscenza.

La rappresentazione della conoscenza in EMYCIN è basata fondamentalmente su regole di produzione; è inoltre provvista una semplice sintassi per specificare gli attributi di un oggetto e le relazioni tra più oggetti.

La premessa delle regole di produzione è costituita da una concatenazione (mediante i connettivi logici AND e NOT) di più predicati, ciascuno dei quali si applica ad una terna oggetto-attributo-valore. La conclusione contiene ancora una terna associativa ed eventualmente un fattore di certezza numerico, che indica il grado in cui si ritiene che la conclusione, data la verità della premessa, sia certa.

Di seguito presentiamo una regola di produzione tratta da MYCIN, espressa nel linguaggio di rappresentazione di EMYCIN e nel suo corrispettivo inglese.

```
PREMISE ($AND (SAME CNTXT INFECT PRIMARY-BACTEREMIA) (MEMBF
                CNTXT SITE STERILESITES)
                (SAME CNTXT PORTAL GI))

ACTION  (CONCLUDE CNTXT IDENT BACTEROIDES TALLY .7)
```

If 1) the infection is primary bacteremia, and
2) the site of the culture is one of the sterilesites, and
3) the suspected portal of entry of the organism is the gastrointestinal tract,

Then there is suggestive evidence (.7) that the identity of the organism is bacteroides.

La regola utilizza i predicati SAME, MEMBF: complessivamente il linguaggio di rappresentazione delle regole mette a disposizione una ventina di predicati, oltre agli ordinari operatori logici e aritmetici.

Gli oggetti a cui si riferiscono i predicati delle regole di produzione sono rappresentati tramite una struttura dati simile al record di un linguaggio di programmazione tradizionale.

Questa struttura dati identifica una lista di attributi dell'oggetto, alcuni dei quali vengono identificati come "GOALS" (obiettivi) della consultazione (il meccanismo inferenziale mira quindi a identificarne il valore) mentre altri sono definiti di tipo "ASK-FIRST" (quando il loro valore è ignoto, il meccanismo inferenziale tenta di accertarlo in primo luogo domandandolo all'utente). Il record descrittivo dell'oggetto identifica inoltre una lista di regole ad esso pertinenti: queste regole referenziano l'oggetto con il nome "CONTEXT" (contesto).

Infine, nello strutturare la base di conoscenza fattuale, un determinato oggetto, o contesto, per usare la terminologia di EMYCIN, è generalmente associato ad altri contesti in una struttura ad albero.

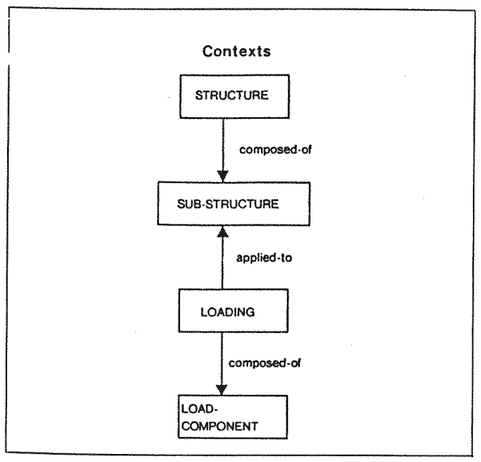


Fig. 1a - L'albero dei contesti di SACON

Nella fig. 1a è illustrato l'albero dei contesti di uno dei sistemi esperti sviluppati con EMYCIN, SACON, un assistente all'impiego di programmi di calcolo strutturale.

La struttura oggetto dell'indagine è composta di più sottostrutture, a ciascuna delle quali è applicato un carico, che ha diverse componenti.

La struttura dati che rappresenta, ad esempio, il carico, referenzierà la sottostruttura come suo antecedente nell'albero dei contesti e la componente di carico come suo discendente (offspring).

2.2 Schema del processo deduttivo.

L'albero dei contesti costituisce in sostanza la spina dorsale della consultazione. Infatti il meccanismo deduttivo di EMYCIN inizia a identificare gli attributi qualificati come obiettivi nel contesto-radice.

A questo scopo esso seleziona un primo obiettivo, e rintraccia nella lista di regole associate al contesto quelle che referenziano nella loro conclusione l'obiettivo prescelto.

Il sistema procede quindi alla valutazione esaustiva delle premesse di ciascuna delle regole pertinenti.
In genere nella premessa di ogni regola è referenziato qualche attributo di valore al momento ignoto; se è così, tale attributo viene assunto come nuovo obiettivo e l'algoritmo si reinnesca ricorsivamente.

Alcune regole possono referenziare attributi di contesti diversi da quello correntemente istanziato; in tal caso l'identificazione di questi attributi comporta l'attivazione di un nuovo contesto, seguendo l'albero statico dei contesti che in tal modo, nel corso della consultazione, genera un albero di istanze derivato dall'albero statico dei contesti. (v. fig. 1b)

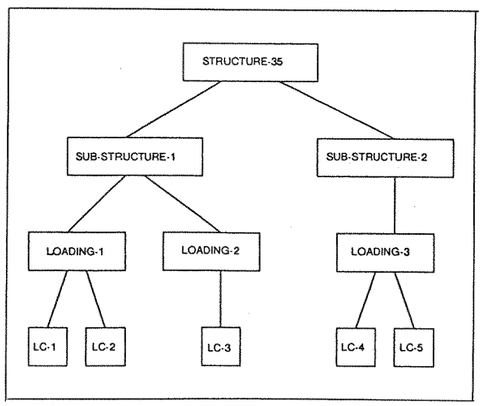


Fig. 1b - Un albero delle istanze in SACON (figura tratta da [1])

Come risulta dalla precedente descrizione sommaria, il meccanismo deduttivo di EMYCIN è un classico esempio di backward chaining: il sistema parte da un obiettivo e risale tramite le regole agli attributi ad esso correlati, sino a determinare attributi il cui valore è - o si presume essere - noto all'utente (quelli che in sede di articolazione della struttura dei contesti sono stati definiti di tipo ASKFIRST), e il cui valore viene pertanto identificato tramite una domanda diretta all'utente.

Come si è premesso, EMYCIN è corredato di una serie di strumenti che hanno la funzione di rendere agevole la creazione, il mantenimento e soprattutto la verifica di una base di conoscenza.

In ogni momento, nel corso di una consultazione, l'utente può chiedere perchè una domanda viene posta (e in risposta il sistema presenta la regola di cui sta effettuando la valutazione) o come una certa conclusione è stata raggiunta. Può inoltre indicare preventivamente una lista di parametri per i quali richiede al sistema di spiegare (EXPLAIN, REVIEW) come vengono attribuiti loro determinati valori, oppure di confrontare tali valori (TEST) con i valori raggiunti in un precedente consultazione, effettuata con un diversa versione della base di conoscenza.

Tutte queste informazioni (come d'altronde le domande) vengono porte all'utente in una forma vicina al linguaggio naturale. Similmente il sistema è in grado di accettare domande espresse in un semplice sottoinsieme del naturale linguaggio naturale riguardo alle conclusioni raggiunte o a caratteristiche generali del dominio di consultazione.

Nella fig. 2 si vedono alcuni esempi di dialogo tratti dai sistemi MYCIN e SACON. Le domande vengono analizzate con tecniche non particolarmente complesse (pattern matching e ricerca di parole chiave in un dizionario fornito dal costruttore della base di conoscenze), ottenendo tuttavia un elevato livello di facilità nell'interazione con il sistema

3. OPS5

OPS5 è il membro più famoso della famiglia di linguaggi OPS (OPS2; OPS4; OPS7), sviluppati presso la Carnegie Mellon University da C. Forgy e altri, [10], [11] in uso ormai da diversi anni per una varietà di applicazioni di Intelligenza Artificiale.

Le caratteristiche di OPS5 lo rendono particolarmente adatto a problemi di rappresentazione della conoscenza mediante l'utilizzo di regole di produzione.

L'applicazione più rilevante di OPS5 nella realizzazione di Sistemi Esperti è senza dubbio R1 [12], un

Examples of general questions about the MYCIN and SACON domains are shown below.

** *IS BACTEROIDES AEROBIC?*

No, the aerobicity of Bacteroides is anaerobic.

** *WHAT ORGANISMS ARE LIKELY TO BE FOUND IN THE THROAT?*

The likely pathoges associated with throat are: DIPLOCOCCUSPNEUMONIAE, STREPTOCOCCUS-GROUP-A, NEISSERIA-MENINGITIDIS.

** *HOW DO YOU USE THE GEOMETRY OF A SUB-STRUCTURE?*

There are 5 rules which use the geometry of the substructure

3, 48, 102, 101, 100

The parameters concluded by these rules are:
the shape of the sub-structure (53 45)
the analysis recommendations to be considered when preparing the structure for modelling (102 101 100)

Which of the rules do you wish to see?
** 53

RULE 053

If:
1) Bucking is one of the deflection phenomena in the structure, and
2) You have examined the sub-structures, and have found one for which the geometry of this sub-structure is curved

Then: The following is one of the analysis recommendations to be considered when preparing the structure for medelling: Buckling estimates should be based on incremental load analysis.

Examples of questions about a particular MYCIN consultation are shown below.

** *IS ORGANISM-1 A STREPTOCOCCUS?*

No. The identity of ORGANISM-1 is E. coli (.7) Bacteroides (.7)
Klebsiella-pneumoniae (.4) Pseudomonas-aeruginosa (.22)
Enterobacter (.21) Proteus-non-mirabilis (.19) Serratia (.03) - Salmonella (.02)

** *WHY DID YOU RULE OUT STREPTOCOCCUS AS A POSSIBILITY FOR ORGANISM-1?*

RULE 033 could have been used to determine that the identity of ORGANISM-1 is streptococcus. However, clause 2 of rule 033 ["the morphology of the organism is coccus"] was already know to be false for ORGANISM-1, so the rule was never tried.

Fig. 2 - Esempi di dialogo tratti dai sistemi MYCIN e SACON

sistema sviluppato da J. Mc Dermott della CMU per la Digital Equipment Corp. per la configurazione di calcolatori Vax, oggi d'uso corrente sulle linee di produzione DEC.

OPS5 è scritto in Franz Lisp operante sotto sistema operativo UNIX vers. 4.2 di Berkeley. CMU ha ritenuto di non adottare alcuna politica commerciale per OPS5: è possibile ottenere gratuitamente il linguaggio senza però alcuna garanzia di mantenimento del prodotto. Viceversa, alcune case di software americane hanno sviluppato loro versioni del linguaggio: tra queste segnaliamo la VERAC inc. che rende disponibile sulla macchina della Symbolics un'estensione di OPS5 denominata OPS5e.

Lo schema di rappresentazione della conoscenza e il meccanismo deduttivo di OPS5 costituiscono un efficiente strumento di base per la costruzione di sistemi basati sulla conoscenza, in quanto non impongono particolari strategie al processo di problem solving, nè particolari vincoli alla rappresentazione. OPS5 consente a chi lo usa di trattare simboli ed esprimere relazioni tra essi; ma nè i simboli nè le relazioni hanno significati predefiniti dal linguaggio: spetta all'utente stabilirli, architettando opportunamente il sistema di regole di produzione.

Preciseremo meglio questo aspetto illustrando gli strumenti forniti da OPS5 per la rappresentazione della conoscenza e il funzionamento del meccanismo deduttivo.

3.1 Struttura dati e regole

OPS5 opera su di una base dati globale denominata memoria di lavoro la quale consiste in un insieme di oggetti simbolici variamente strutturati. La struttura d'impiego più comune consente di rappresentare un oggetto associandogli un insieme di coppie attributo-valore. Ad esempio, l'elemento "blocco, identificato come block1, di colore rosso, che pesa 500 gr. e misura 100 mm. per lato", può essere rappresentato nella memoria di lavoro con l'oggetto della fig. 3a

Durante l'esecuzione del programma la definizione di un oggetto nella memoria di lavoro può variare non soltanto perchè possono variare i valori degli at-

```
(block
        ∧ name block1
        ∧ color red
        ∧ mass 500
        ∧ length 100
        ∧ width 100
        ∧ height 100)
```

Fig. 3a

```
(p find-colored-block
(goal                          ;se è presente un goal
    ∧ status active            ;attivo
    ∧ type                     ;per trovare
    ∧ object block             ;un blocco
    ∧ color ‹z›      )         ;di un certo colore
(block                         ;ed il blocco
    ∧ color ‹z›                ;di quel colore esiste
    ∧ name ‹block›)

⇒                              ;allora
(make result                   ;genera un elemento
    ∧ pointer ‹block›)         ;che punti al blocco
(modify 1                      ;e modifica il goal
    ∧ status satisfied))       ;indicandolo soddisfatto.
```

Fig. 3b

tributi, ma anche perchè è possibile cancellare o aggiungere coppie attributivo-valore. Gli oggetti della memoria di lavoro vengono creati, modificati, cancellati dall'insieme di regole che costituiscono la "production memory".

Queste regole hanno la classifica forma di produzioni:

‹antecedente› ⇒ ‹conseguente›

Di seguito (fig. 3b) viene riportata una tipica (sebbene alquanto semplice) produzione espressa in OPS5.

L'antecedente, o parte sinistra, della regola è costituito da un insieme di oggetti denominati "condition elements". Nella produzione di fig. 3b "goal" e "block" sono i due "condition elements" dell'antecedente. Il "condition element" è del tutto uguale ad un oggetto rappresentato nella memoria di lavoro, eccetto per il fatto che i valori dei suoi attributi possono essere variabili.

es.: (block ∧ color ‹z›
∧ name ‹block›)

‹z› e ‹block› rappresentano due valori variabili.

Una produzione può essere eseguita quando tutti i "condition element" definiti nell'antecedente corrispondono con gli oggetti presenti nella memoria di lavoro in quel momento.

La verifica di un "condition element" avviene confrontando i valori dei suoi attributi con quelli di un oggetto dello stesso tipo presente nella memoria di lavoro, secondo le modalità di seguito illustrate:

– se ad un attributo è associato un valore costante

(simbolico o numerico), questo trova corrispondenza solo in un attributo avente lo stesso valore, ad esempio l'oggetto:

(block ∧ color red ∧ mass 500)

corrisponde all'oggetto di tipo block che abbiamo definito in fig. 3a in quanto corrispondono i valori degli attributi "color" e "mass".

– Se ad un attributo è associato un valore variabile, questo trova corrispondenza con il valore (simbolico o numerico), qualsiasi esso sia, associato a quell'attributo nell'elemento della memoria di lavoro. Se, però, la variabile è referenziata più volte nella stessa regola (associata ad attributi di altri "condition element") tutte le sue istanze devono assumere lo stesso valore. In questo caso il "condition element" si dice correlato all'oggetto della memoria di lavoro in grado di verificare i suoi valori variabili.

  Rifacendosi alla produzione illustrata in fig. 3b e ammettendo che nella memoria di lavoro esista l'oggetto "block" definito, in fig. 3a abbiamo che il "condition element"

  (block ∧ color ‹z› ∧ name ‹block›),

  corrisponde all'oggetto della memoria di lavoro, correlando le variabili ‹z› e ‹block› rispettivamente ai simboli "red" e "block1". La variabile ‹z› risulta correlata a "red" anche nell'altro "condition element" ("goal") della produzione.

L'esecuzione di una produzione consiste nell'attuare le azioni specificate nel conseguente, o parte destra, della regola. Tra queste le più significative sono quelle che specificano operazioni da compiersi sulla memoria di lavoro; le azioni principali sono:

– *make:* crea un nuovo oggetto nella memoria di lavoro.

  Ad esempio, riferendoci sempre alla produzione indicata, supponendo che dalla valutazione dell'antecedente risulti che la variabile ‹block› sia correlata a "block1"; allora l'azione

  Es.:
  (make result ∧ pointer ‹block›)

  crea l'elemento (result ∧ pointer block1) nella memoria di lavoro.

– *remove:* cancella uno o più oggetti dalla memoria di lavoro.

  Es.:
  (remove 1)

  cancella dalla memoria di lavoro l'oggetto che corrisponde al primo "condition element" dell'antecedente. (Nella produzione di fig. 3b l'oggetto goal").

– *modify:* modifica i valori di uno o più attributi di un oggetto nella memoria di lavoro.
  Facendo riferimento alla produzione illustrata di fig. 3b l'azione:

  (modify 1 ∧ status satisfied)

  cambia il valore dell'attributo "status" relativo all'oggetto che verifica il primo "condition element" (goal) della parte sinistra della regola. In questo caso l'oggetto presente nella memoria di lavoro diventa:

  (goal ∧ status satisfied ∧ type find ∧ object block...)

  Esiste inoltre un insieme di azioni che permettono di gestire l'input/output da files, aprendo e chiudendo files e leggendo o scrivendo elementi su tali files.

3.2 Struttura di controllo.

L'interprete OPS5 processa un insieme di produzioni compiendo ciclicamente un sequenza di operazioni denominata "riconoscimento-azione", che consiste nei seguenti passi:

1. *MATCH.* Valutazione dell'antecedente di ogni produzione per determinare quali produzioni soddisfano allo stato corrente della memoria di lavoro.

2. *CONFLICT RESOLUTION.* Selezione di una delle produzioni individuate al fine di eseguirla. La selezione viene operata in base a semplici criteri, ad esempio scegliendo la regola più specifica, cioè quella che referenzia nell'antecedente il maggior numero di oggetti. Se nessuna produzione soddisfa lo stato della memoria di lavoro, la sessione termina.

3. *ACT.* Esecuzione della parte destra della regola.

4. *LOOP.* Riesecuzione dal punto 1.

La sequenza riconoscimento-azione, nella sua semplicità, non introduce particolari strategie di controllo; l'utente stesso, appoggiandosi a questo semplice meccanismo, può realizzare le strategie che si adattano alla risoluzione del proprio problema.

Ad esempio, ciò può avvenire attraverso la definizione esplicita di "goals", ovvero elementi della memoria di lavoro che specificano l'azione da perseguire nei cicli successivi. Le regole congruenti con il goal da perseguire lo referenziano nel loro antecedente, in modo che tali regole possono essere attivate solo quando un goal di quel tipo è stato istanziato nella memoria di lavoro. L'evolversi del sistema a produzioni viene quindi controllato inserendo in sequenza goals diversi nella memoria di lavoro e rimuovendoli quando sono stati perseguiti, ossia quando i valori dei loro attributi sono stati specificati.

3.3 Conclusioni

A differenza di EMYCIN, e di altri sistemi di sviluppo, OPS5 non dispone di sofisticati strumenti di interfaccia utente per la creazione della base di conoscenza e la spiegazione del processo di ragionamento.

Il suo punto di forza risiede nella sua generalità, che consente a chi lo impiega di realizzare strutture dati e meccanismi di controllo diversificati, per far fronte alle proprie esigenze applicative.

Particolarmente sofisticato è invece l'algoritmo di pattern matching dell'interprete, che è un aspetto di grande importanza ai fini dell'efficienza, essendo stimato che il 90% del tempo di esecuzione in applicazioni basate sulla conoscenza venga inpiegato nel matching.

Il punto debole di OPS5 è la mancanza di un sofisticato ambiente di programmazione: l'interprete fornisce alcuni semplici ausili al debugging, a livello comunque molto inferiore a funzioni quali "HOW" e "WHY" in Emycin.

4. EXPERT

EXPERT [7], [8] è un sistema di sviluppo per sistemi esperti di tipo consultativo, realizzato presso l'università di Rutgers sotto la guida di S. Weiss e C. Kulikowsky. EXPERT è stato sviluppato generalizzando i concetti e l'architettura di CASNET [6], un modello consultativo per la diagnosi e la cura del glaucoma, realizzato dagli stessi autori di EXPERT.

Fino ad oggi EXPERT è stato utilizzato per lo sviluppo di sistemi esperti in diversi campi di elaborazione, tra cui la medicina, la chimica (esami di laboratorio) [14], la geologia [13].

EXPERT è scritto in Fortran ed è disponibile su VAX con sistema operativo VMS e su IBM con sistema operativo VM/CMS. Il sistema è commercializzato e supportato dalla Rutgers University stessa.

4.1 Rappresentazione della conoscenza.

La rappresentazione della conoscenza in EXPERT è basata su tre componenti: le *ipotesi,* i *fatti* e le *regole di decisione* che correlano fatti ed ipotesi.

Le *ipotesi* sono le conclusioni che il sistema deduce, ad esempio: diagnosi, indicazioni di terapie.

Per *fatti* si intendono quei dati osservabili che sono rilevanti al fine del raggiungimento di una conclusione: essi sono valutati dall'utente su richiesta del sistema. In una applicazione in ambiente medico, ad esempio, sono fatti i sintomi presentati dal paziente e i risultati dei test di laboratorio.

Le *regole di decisione* hanno la consueta forma di regole di produzione. Esse consentono di correlare in vario modo fatti ed ipotesi.

È caratteristica in EXPERT la presenza di diversi meccanismi che consentono al progettista della base di conoscenza di indirizzare lo svolgimento della consultazione (stima della frequenza di occorrenza delle ipotesi; possibilità di ordinare le ipotesi in una tassonomia; questionari; fattori di costo). Nelle sezioni che seguono presentiamo con qualche dettaglio la sintassi della rappresentazione della conoscenza in EXPERT, cercando di sottolineare la funzione e la ragione d'essere di questi meccanismi.

4.1.1 Ipotesi

La sezione dichiarativa delle ipotesi (la prima ad essere dichiarata nel modello di rappresentazione, identificata dalla parola chiave **HYPOTHESES) porta alla definizione di una rete tassonomico-causale, i cui nodi terminali sono costituiti dalle possibili conclusioni diagnostiche, o ipotesi finali, mentre i nodi intermedi rappresentano categorie diagnostiche, o ipotesi parziali, a ciascuna delle quali viene associato un peso che rappresenta la frequenza di occorrenza che il progettista del modello causale le attribuisce a priori nell'ambito della conclusione a cui si riferisce.

Riportiamo un esempio che illustra l'impiego del costrutto sintattico TAXONOMY, che permette di strutturare la rete causale:

* TAXONOMY

CWS	L'auto non parte
FUEL	.Problemi di alimentazione (.5)
FLOD	..Auto ingolfata (.2)
CHOK	..Rottura della pompa (.2)
EMPT	..Non c'è benzina (.2)
FILT	..Filtro del carburante intasato (.2)
ELEC	.Problemi all'impianto elettrico (.5)
CAB	..Cavi batteria sconnessi o corrosi (.2)
BATD	..Batteria scarica (.2)
STRT	..Motore d'avviamento non funziona (.2)

Come si vede, ogni ipotesi è identificata da un mnemonico, da un certo numero di punti che indica la sua posizione gerarchica nella tassonomia, da una linea di testo utilizzata dal sistema nell'interazione con l'utente, e da un peso.

La tassonomia viene interpretata, in sede di esecuzione, come una rete causale, nella quale il peso associato ad ogni ramo è assunto come la certezza a priori (cioè indipendente dalla valutazione effettuata a run time) della relazione di causa tra i due nodi che il ramo collega. Tale peso concorre a determinare, in sede di consultazione, il fattore di certezza assegnato alle ipotesi via via considerate.

Va osservato che non è necessario esprimere relazioni tassonomiche tra le ipotesi. La forma più semplice di un modello può consistere in una semplice lista di ipotesi finali.

La sezione dichiarativa HYPOTHESES può includere, opzionalmente, anche una lista di ipotesi intermedie, che vengono dichiarate allo scopo di referenziare in più regole, con un solo identificatore, un concorso di cause che viene in effetti concettualizzato come un unico fattore. Con riferimento al precedente esempio, potendoci essere più problemi relativi alla batteria (CAB, BATD), si può creare una ipotesi intermedia BATT per dire in modo sommario che la batteria non funziona:

```
* INTERMEDIATE HYPOTESES

BATT    Problemi alla batteria
CAB     .Cavi batteria sconnessi o corrosi
BATD    .Batteria scarica
```

Infine, con notazione corrispondente a quella delle precedenti due sezioni, si possono descrivere le terapie previste (TREATMENTS).

```
*TREATMENTS

REP     .Riparazioni auto
WAIT    .Attendi 10 minuti o premi l'acceleratore
          a fondo partendo.
OPEN    .Rimuovi il filtro aria e apri la pompa ben-
          zina.
GAS     .Introduci benzina nel serbatoio
RFLT    .Cambia filtro carburante
CLEN    .Pulisci e stringi i morsetti della batteria.
GBAT    .Carica o rimpiazza la batteria.
NSTR    .Cambia il motorino d'avviamento.
```

Nella tassonomia TREATMENTS le terapie di più alto livello corrispondono alle ipotesi finali, mentre gli altri nodi corrispondono a sottoinsiemi di possibili terapie, e i pesi, se usati, indicano la confidenza a priori nel successo di una particolare terapia.

Come vedremo in seguito, la corrispondenza tra conclusioni diagnostiche e terapie viene stabilita tramite regole HH.

4.1.2 Fatti

I fatti ("findings") sono le informazioni fornite dall'utente durante la consultazione. Essi sono rappresentati come attributi degli oggetti della consultazione, che possono essere presenti, assenti o non determinabili; oppure come variabili numeriche che assumono un valore entro un intervallo prespecificato. Si noti che non vi è alcun costrutto che specifichi che un insieme di attributi è relativo allo stesso oggetto, come ad esempio avviene in EMYCIN.

Nei casi in cui sia richiesto di esprimere l'incertezza associata al risultato di una osservazione, essa deve essere rappresentata esplicitamente come una variabile addizionale.

Ad esempio, l'accuratezza del risultato di un esame di laboratorio può essere richiesta esplicitamente scrivendo opportune regole di decisione per modificare le inferenze relative al risultato del test. Si mette dunque a disposizione del progettista uno strumento a basso livello piuttosto che una politica generalizzata per gestire l'incertezza, in coerenza con l'impostazione generale di EXPERT.

Nella costruzione del modello i fatti vengono introdotti nella sezione **FINDINGS (che appare dopo la selezione **HYPOTHESES) specificando il tipo della corrispondente domanda da porre all'utente. Esistono quattro tipi di domande:

1. MULTIPLE CHOICE:

i fatti presentati sono mutuamente esclusivi, solo uno può essere scelto.

Nel modello questo tipo di domanda viene definito nel seguente modo:

```
*Multiple choice
Odore di carburante nella vettura
HGAS    non presente
MGAS    debole
LGAS    molto forte
```

Tale domanda, durante la consultazione, comparirà nel seguente formato:

```
Odore di carburante nella vettura
1)  non presente
2)  normale
3)  molto forte
Choose one.
*
```


Gli identificatori che sintetizzano gli oggetti, che non vengono visualizzati in fase di presentazione della domanda, compaiono, insieme agli identificatori della HYPOTHESES, nella rappresentazione delle regole.

2. CHECKLIST:

i fatti presentati non sono mutamente esclusivi, se ne può scegliere anche più d'uno.

Nel modello questo tipo di scelta viene definito come segue:

```
* Checklist
Luci posteriori non funzionanti
PHEAD   luci di posizione
PSTOP   luci di stop
PTURN   indicatori di direzione
```

Durante la consultazione la domanda viene presentata come segue:

```
Luci posteriori non funzionanti:
1)  luci di posizione
2)  luci di stop
3)  indicatori di direzione
Checklist
*
```

3. NUMERICAL:

al fatto prospettato deve essere assegnato un valore entro i limiti stabiliti.

Nel modello questa domanda è definita nel seguente modo:

```
*NUMERICAL /MIN=0 /MAX=100/;
TWAT temperatura dell'acqua (°C):
```

A tempo di esecuzione viene presentata nella forma:

```
temperatura dell'acqua (°C):
*
```

4. YES/NO:

domande a cui può essere risposto solo affermativamente o negativamente.

Nel modello questa domanda è definita nel seguente modo:

```
*YES-NO
EGAS indicatore carburante segnala vuoto
```

A tempo di esecuzione viene presentata nella forma:

```
indicatore carburante segnala vuoto
*
```

All'interno della sezione FINDINGS esiste la possibilità di utilizzare una struttura di controllo per influire sulla sequenza con cui il sistema richiede all'utente i fatti. Essa consente di raggruppare insieme diverse domande per formare un questionario.

Quando il meccanismo di controllo richiede un fatto incluso in una domanda facente parte di un questionario, allora tutte le domande relative al questionario devono essere poste nell'ordine in cui compaiono nel testo.

Di seguito è illustrato un esempio di questionario:

```
**FINDINGS
    . . .
    . . .

*BEGIN QUESTIONNAIRE

*Checklist
Luci dell'auto non funzionanti
FRONT   Luci anteriori
REAR    Luci posteriori

*Checklist
Luci anteriori non funzionanti
AHEAD   Luci di posiizone
ATURN   Indicatori di direzione

*Checklist
Luci posteriori non funzionanti
PHEAD   Luci di posizione
PSTOP   Luci di stop
PTURN   Indicatori di direzione

*END QUESTIONNAIRE
```

A ciascuna domanda può essere associato un *costo* relativo all'acquisizione del dato che fornisce. Specificare il costo di una richiesta è importante per l'ordine in cui si vogliono acquisire i fatti; infatti il meccanismo di controllo cerca sempre di valutare per primi i fatti con costo minore, in modo tale che i test più costosi (o più rischiosi) siano richiesti solo nel caso in cui le informazioni a disposizione non siano sufficienti a risolvere il problema.

4.1.3 Regole di decisione

Vi sono tre tipi di regole per descrivere le relazioni logiche tra fatti ed ipotesi:

- *regole FF:* correlano fatti ad altri fatti;
- *regole FH:* correlano fatti ad ipotesi;
- *regole HH:* correlano ipotesi ad altre ipotesi

Le regole FF hanno in sostanza la funzione di derivare delle risposte implicite sul valore di un certo fatto in dipendenza da un altro fatto asserito esplicitament. Esse sono particolarmente utili per esprimere salti condizionali nell'ordine di scansione di un questionario.

Facendo riferimento al questionario introdotto precedentemente, possiamo definire le seguenti regole FF:

F(FRONT,F) ⇒ F(AHEAD:ATURN,F)
F(READ,F) ⇒ F(PHEAD:PTURN,F)

Dove la notazione AHEAD : ATURN sta ad indicare tutti i fatti compresi tra AHEAD e ATURN, estremi inclusi.

Le regole FH esprimono combinazioni logiche di fatti che confermano o negano, con un certo fattore di certezza, una determinata ipotesi.
Esempio:

F(TEMP,0:50)& [1:F(SCRN,T),F(OCRN,T)] ⇒ H(CHOK, 0.6)

"Se la temperatura (TEMP) è nel range 0-50°C e il motore d'avviamento funziona regolarmente (OCRN) oppure lentamente (SCRN), vi è la possibilità (0.6) che la pompa della benzina malfunzioni (CHOK)".

Ogni regola HH è costituita da due parti distinte: la parte "if" o contesto, consistente in un predicato che funge da condizione preliminare della regola; e la parte "then", o tavola di regole, consistente in un insieme di relazioni che vengono valutate solo se il contesto è verificato. Sia la premessa delle relazioni che compaiono nella tavola di regole che il contesto sono costituiti da combinazioni di ipotesi e fatti.

*HH RULES
*IF
F(FCWS,T)
"Se la macchina non parte" "contesto"

*THEN "Tavola di regole"

H(FLOD,0.2:1) ⇒ H(WAIT,0.9)
"Se la macchina è ingolfata (ipotesi FLOD), attendi 10 min. (trattamento WAIT)".

F(INTC,T)&H(DBAT,.2:1) ⇒ H(DYN,0.9)
"Se la cinghia di trasmissione è integra (ipotesi INTC) e si ritiene che la batteria sia scarica (ipotesi DBAT) può essere guasta la dinamo (ipotesi DYN)".

*END

Si noti che le ipotesi che figurano nel contesto o in una premessa per essere verificate devono essere associate ad un fattore di incertezza entro un prescritto intervallo. Tali fattori di certezza sono quelli stabiliti direttamente da qualche altra regola FH o HH.

Le regole sono valutate nell'ordine in cui sono dichiarate, per cui se una regola HH implica una ipotesi H1 che risulta necessaria per stabilire una ulteriore ipotesi H2, essa deve figurare prima nel file di dichiarazione.

4.2 Meccanismo di controllo

L'obiettivo principale nel progetto dell'algoritmo di controllo di EXPERT è stato quello di arrivare a disporre di un meccanismo deduttivo più semplice ed efficiente possibile, ma in grado nel contempo di fornire al progettista del modello causale la possibilità di controllare efficacemente la strategia di selezione delle domande e di verificare la consistenza delle ipotesi.

A differenza di EMYCIN il meccanismo di valutazione delle regole di EXPERT non agisce "backward-chaining"; infatti le regole in EXPERT sono valutate ed eseguite in modo preordinato, strettamente correlato alla struttura che il progettista ha attribuito al modello causale.

D'altro lato, l'dentificazione del prossimo fatto da accertare, all'inizio di un nuovo ciclo di valutazione, viene effettuato in base ad una strategia che ha l'obiettivo di attivare quelle regole che possono confermare ulteriormente l'ipotesi che al momento della scelta risulta più probabile. L'utente può tuttavia sospendere questa strategia di scelta, non sempre intuitiva, facendo uso del costrutto QUESTIONNAIRE nella sezione **FINDINGS, che, come è stato precedentemente osservato, fa sì che i fatti elencati nel questionario vengano richiesti in sequenza.

Ogni volta che l'utente asserisce un nuovo fatto, l'algoritmo di deduzione esegue questa sequenza di operazioni: dapprima vengono valutate le regole FF, per accertare l'esistenza di fatti che discendono direttamente dal fatto asserito, quindi vengono valutate le regole FH che contengono nella premessa il fatto in questione e quelli eventualmente derivati da esso tramite le regole FF; infine si valutano le regole HH.

Per ognuna di esse si considera prima la parte "if" di ogni tavola di regole: quindi le relazioni contenute nelle tavole attive sono valutate e – quando verificano – eseguite. Si noti che, nel corso di questa valutazione, le regole HH possono influenzarsi tra loro, in quanto ipotesi asserite con un certo coefficiente di certezza in una regola possono essere presenti nella premessa di altre regole sicchè, dato che le regole sono valutate nella sequenza in cui sono state introdotte nel modello, il loro ordine relativo risulta importante.

Al termine della sequenza di operazioni sopra descritte il sistema attribuisce alle ipotesi provate un fattore di certezza che viene valutato come segue: nel caso in cui un'ipotesi sia implicata da più di una regola le viene associato il coefficiente di certezza massimo in valore assoluto; i fattori di certezza così ottenuti vengono quindi propagati secondo la rete causale ottenua dalla tassonomia della ipotesi, tenendo conto in tal modo delle correlazioni statiche tra le ipotesi, definite a priori dal modello.

Non riportiamo qui per brevità l'algoritmo adottato per calcolare la propagazione dei fattori di certezza, che gli autori di EXPERT hanno sviluppato nel corso della loro esperienza su CASNET; si confronti il riferimento [6].

Infine ai fattori di certezza viene applicata un'ulteriore correzione al fine di avvantaggiare le ipotesi con il maggior numero di regole verificate che le asseriscono.

I fattori di certezza così ottenuti governano le operazioni del meccanismo inferenziale nella scelta del prossimo fatto da accertare, all'inizio del sucessivo ciclo di valutazione.

Fra le regole (FH o HH) non ancora attivate, vengono infatti prescelte le regole che asseriscono le ipotesi al momento parzialmente verificate (a cui cioè è associato un fattore di certezza positivo) con un fattore di certezza maggiore di quello attualmente associato alle predette ipotesi.

Il fatto da richiedere è quindi selezionato tra quelli che compaiono nelle regole così individuate in base ai seguenti criteri:

- il costo associato al fatto sia minimo;
- il fatto concorra alla valutazione di una regola che implica l'ipotesi più probabile;
- il fatto concorra alla valutazione di regole che aumentino (in valore assoluto) il fattore di certezza dell'ipotesi.

4.3 Considerazioni conclusive

Le operazioni di scrittura-esecuzione di un modello di consultazione tramite EXPERT sono in certo modo simili a quelle di un programma tradizionale.

Per la scrittura del modello si utilizza un normale text-editor col quale si genera un file scritto nel linguaggio di rappresentazione della conoscenza. Il modello così costruito viene processato da un primo modulo che esegue una verifica sulla presenza di errori sintattici e genera un'efficiente versione compilata della base di conoscenza introdotta.

Su di esso agisce un secondo modulo che è il vero e proprio programma di consultazione che, oltre alla funzionalità descritta nel paragrafo precedente, mette a disposizione diverse facilitazioni per quel che riguarda l'interfaccia verso l'utente. Ad esempio il sistema conclude una consultazione stampando un sommario costituito dal riepilogo delle risposte relative ai fatti che sono stati richiesti durante la consultazione, e dall'interpretazione del caso che il sistema ha fornito, consistente nelle ipotesi che risultano accertate e negli eventuali trattamenti raccomandati, con il relativo fattore di certezza.

Oltre a ciò il sistema è in grado di fornire spiegazioni sulle deduzioni eseguite, sottolineando i fatti e le ipotesi intermedie che concorrono all'asserzione di un'ipotesi.
Altre facilitazioni riguardano la possibilità di ottenere deduzioni parziali prima che la consultazione sia terminata, sulla base dei dati introdotti fino a quel momento, e di rimodificare dati precedentemente introdotti.

Il pregio principale di EXPERT risiede senza dubbio nella sua facilità d'uso. Grazie ad essa è possibile sviluppare rapidamente un primo prototipo in grado di funzionare. Le diverse facilitazioni a livello di interfaccia utente e l'efficienza sia del modulo di compilazion che di quello di esecuzione consentono all'utente di EXPERT di procedere speditamente nel test del modello, secondo il ciclo di accrescimento-test-modifica proprio dei sistemi esperti.

5. SAGE

Nell'ambito degli "skeletal systems" SAGE [9] rappresenta il primo esempio di strumento appositamente realizzato per essere commercializzato ed impiegato in applicazioni industriali. Questa sua caratteristica di "prodotto" commerciale nato al di fuori di una qualsiasi applicazione specifica, gli merita un esame anche se necessariamene breve ed incompleto.

Vale la pena di notare che la causa prima di tale incompletezza risiede nella "giovinezza" di SAGE: la prima versione disponibile è infatti della seconda metà del 1982.

Al momento in cui queste note vengono scritte e per quanto ci risulta, l'unico Expert System realizzato facendo uso di SAGE è X-DOC, un prototipo di sistema esperto dedicato ad assistere l'utente in un processo decisionale e prodotto dalla stessa casa produttrice di SAGE (SPL International) a scopo dimostrativo. Non è quindi possibile disporre di alcun parametro di giudizio sulla efficienza di SAGE al mo-

mento del suo impiego per una determinata applicazione.

5.1 La struttura di SAGE

SAGE si compone di due sottosistemi: il compilatore e l'executive. Il primo con la funzione di fornire gli strumenti per la codifica della base di conoscenza (modello) e principalmente un apposito linguaggio. Il secondo con lo scopo primario di guidare ed assistere l'utente nella consultazione del modello stesso.

SAGE è disponibile su macchine e sotto sistemi operativi diversi: su VAX sotto VMS 3.0, su PDP11, su LSI11-23 e su IMB PC.

Nel seguito, e conformemente ai limiti di questa presentazione, si descriveranno solo alcuni dei principali costrutti linguistici proposti da SAGE per la codifica del modello e si fornirà un breve elenco delle funzioni più importanti offerte all'utente dall'executive.

5.2 Modalità di rappresentazione della conoscenza.

Per introdurre alcuni dei principali costrutti linguistici offerti da SAGE per la rappresentazione della conoscenza di farà ricorso a semplici esempi che hanno un valore puramente indicativo e non pretendono di essere in alcun modo completi.

In questi esempi, le parole chiave del linguaggio sono definite mediante l'uso di lettere maiuscole mentre in minuscolo sono riportati tutti i nomi definiti dall'utente.

Le frasi racchiuse fra virgolette (") sono anch'esse definite a cura dell'utente e vengono usate dall'executive SAGE al momento di porre domande, di fornire spiegazioni ecc.

Un uso attento di queste frasi è particolarmente consigliato per aumentare la comprensibilità globale del modello compilato tramite SAGE, sia durante la sua messa a punto che durante la successiva consultazione.

- **Il concetto di AREA.**

  La base di conoscenza, in SAGE, può essere divisa in differenti "aree". Ognuna di queste aree costituisce una zona indipendente di investigazione nel senso che una nuova area del modello sarà presa in considerazione solo quando quella in esame sarà stata esaurita.
  I risultati emersi dall'esame di una data area sono però disponibili per quelle successive come pure gli elementi che la compongono sono visibili a tutte le altre. È possibile, inoltre, che per completare l'esame di un'area si proceda alla valutazione completa di un'altra.

- **Gli elementi base del modello: ASSERTION, OBJECT.**

  In SAGE si hanno a disposizione due costrutti linguistici principali per descrivere gli elementi base del modello: ASSERTION e OBJECT. Il primo costrutto è usato per definire delle affermazioni e reca associato un valore (numero razionale compreso fra –100 e +100) che esprime il "grado di credito" o "verosimiglianza" della affermazione stessa.

  Sul legame esistente fra questo valore e la verosimiglianza dell'affermazione si tornerà nel seguito. Qui importa sottolineare che questo valore può essere associato a priori ad una affermazione (DEFAULT) e che può essere mutato nel corso della consultazione del modello o direttamente dall'utente (ASKABLE) o per l'applicazione di determinate regole.

  Come esempio si consideri la seguente ASSERTION che definisce se un certo soggetto è o meno sovrappeso e che viene fatta stimare dall'utente in base a propri elementi di giudizio:

```
ASSERTION sovrappeso: "il soggetto è sopra il
                       peso forma"
DEFAULT 0.0
ASKABLE
```

  Il fatto che questa ASSERTION sia ASKABLE comporterà che, al momento della sua valutazione, l'utente si vedrà porre una domanda del tipo:

  "How certain are you that il soggetto è sopra il peso forma?

  Il bilinguismo dell'esempio, che può apparire alquanto ridicolo, ha la funzione di indicare chiaramente la parte del messaggio direttamente prodotta da SAGE e la parte definita nella "string" associata alla ASSERTION. Il valore 0.0 presente nel campo DEFAULT indica che non esistono motivi a priori per ritenere credibile o meno tale affermazione.

  Il concetto di "credito", nel senso di "stima della verosimiglianza" di una affermazione, merita di essere brevemente precisato, cosa che sarà fatta nel seguito.

  Il secondo costrutto che si vuole presentare è l'OBJECT. Questo permette di definire un elemento del modello a cui viene associato un numero razionale che ne rappresenta il valore e non più, come nel caso dell'ASSERTION, la stima di verosimiglianza.

  I valori che possono essere associati ad un elemento non saranno più, quindi, soggetti al vincolo di essere compresi fra –100 e +100.

  A titolo esemplificativo si riportano alcune definizioni di OBJECT che definiscono elementi cui si ricorrerà nei prossimi esempi. I numeri fra parentesi rappresentano i limiti di accettabilità del valore associato all'elemento.

  Anche in questo caso, la presenza dell'opzione ASKABLE indica che il valore verrà richiesto all'utente. Si noti che, a differenza di quanto avviene per una ASSERTION, la domanda viene formulata usando solo la frase definita nella "string" mentre viene omessa la parte "How certain are you that".

```
OBJECT contenuto calorico: "Valore in calorie
alimentari di una dieta tipo del soggetto considera-
to".
(0.0 , max-cal)

OBJECT pasta: "Quanta pasta in un pranzo tipo?
(in gr.)"
(0.0 , 1000.0)
ASKABLE

OBJECT ragù: "Quanto ragù sulla pasta? (in gr.)"
(0.0 , 100.0)
ASKABLE
```

  A conclusione di questo paragrafo, si fa notare che SAGE consente la definizione di elementi costanti (CONSTANT), si veda negli esempi "max-cal", la cui definizione è abbastanza ovvia da non richiedere alcuna presentazione particolare.

I criteri di valutazione di SAGE

  Nella valutazione del modello, SAGE è in grado di ricorrere a diversi criteri che dipendono dal tipo di elemento considerato.

  Nel caso di ASSERTION viene valutato il grado di verosimiglianza di una certa affermazione mediante una funzione che lo lega alla probabilità della affermazione stessa. La funzione è:

$$\text{verosomiglianza} = \begin{cases} -100.0 & 0 \leq + p < 10^{-10} \\ 10 \log_{10} (p/(1\text{-}p)) & 10^{-10} < p < 1\text{-}10^{-10} \\ +100.0 & 1\text{-}10^{10} < p < 1 \end{cases}$$

  con p = probabilità

  L'utente al quale viene richiesto di valutare la verosimiglianza di una data affermazione ha a disposizione la scala dei razionali compresa fra –5.0 e +5.0. Detto "c" il valore prescelto, la verosimiglianza viene computata nel seguente modo:

$$\text{verosomiglianza} = \log_{10} \frac{(5 + C)}{(5 - C)}$$

  Con queste trasformazioni si ottiene l'effetto di estendere gli estremi della scala avvicinandosi maggiormente al modo di ragionare dell'uomo rispetto ad una scala lineare.

  ASSERTION, OBJECT e CONSTANT possono essere combinati come elementi di valutazione ricorrendo alla così detta "Fuzzy logic" che estende la definizione degli operatori AND, OR, NOT, EXCLOR delle variabili di tipo booleano ai numeri razionali nel seguente modo:

  detti x e y due numeri razionali compresi fra –100.0 e +100.0, si ha:

```
AND: x AND y da il minimo fra x ed y;
OR: x OR y da il massimo fra x ed y;
NOT: NOT x da -x;
EXCLOR: x EXCLOR y è (x AND NOT y) or
(y AND NOT x);
```

  Infine, OBJECT e CONSTANT possono essere manipolati mediante i consueti operatori aritmetici.

- **I passi dell'indagine: ACTION**

  Definiti gli elementi del modello, all'interno di ogni area di indagine, SAGE introduce il costrutto linguistico ACTION che consente all'utente di fissare gli obiettivi finali e parziali dell'esame cioè i passi in cui questo si articolerà.

  A titolo esemplificativo, supponiamo di dover esprimere un passo d'indagine di questo tipo:

  "Ammesso che il soggetto sia sovrappeso, si esamini il contenuto calorico di un suo pasto tipo e si consigli comunque di perdere il peso superfluo eliminando un certo alimento".

  Si suppone che: per poter stabilire l'alimento da eliminare dalla dieta si definisce un'area d'indagine a se stante chiamata "alimento-sconsigliato" e che dall'esame di questa si ricava un OBJECT, "no-alimento-sconsigliato", in grado di qualificare con un numero d'ordine l'alimento.

  Inoltre, si definisce un OBJECT, "contributo-calorico", che rappresenta il totale delle calorie alimentari di un pasto tipo ed, infine, si definisce una ASSERTION, "sovrappeso", che rappresenta una stima del fatto che il soggetto sia sopra il peso forma. In SAGE il passo d'indagine si esprime così:

  ACTION nome-dell'azione: "Si esamina il quarto

passo dell'indagine"

CONSIDER contenuto-calorico, alimento-sconsigliato

ALSO

ADVISE "Si consiglia vivamente di perdere il peso superfluo eliminando l'alimento numero" no-alimento-sconsigliato

PROVIDED sovrappeso

Si noti come la clausola: "Ammesso che il soggetto sia sovrappeso" viene espressa da SAGE mediante la specificazione "PROVIDED nome-della-clausola". Questo impiego della specificazione PROVIDED fa si che la clausola (che può essere un'espressione comunque complessa che lega fra di loro vari elementi) sia valutata prima della ACTION cui si riferisce. Se il valore della clausola è maggiore o uguale a 0.0 la ACTION viene eseguita, altrimenti no.

- **L'espressione delle regole**

Proseguendo nell'esame di questo semplice esempio, bisogna considerare il modo di determinare il valore di "contenuto-calorico" e di guidare altresì l'utente nella valutazione del fatto se sia o meno sovrappeso nel caso in cui non esprima abbastanza certezza di esserlo. Fissiamo +1.0 come valore di certezza sufficiente.

Ciò viene fatto in SAGE definendo delle opportune regole mediante il costrutto linguistico RULE.

Cominciamo considerando il caso di "contenuto-calorico" definito mediante una funzione aritmetica che valuta la quantità ed il contenuto calorico specifico di alcuni alimenti tipici di un pasto. Gli alimenti qui usati e non definiti in precedenti esempi, sono da considerarsi delle CONSTANT. In SAGE si scriverà:

RULE nome-della-regola: "Stima del contenuto calorico"
contenuto-calorico IS
pasta * calorie-pasta + sugo * calorie-sugo + burro * calorie-burro ecc. ecc.

E ancora possiamo voler esprimere la regola:

"se l'utente non si è dimostrato abbastanza sicuro di non essere in sovrappeso, questo fatto viene stimato valutandone alcune caratteristiche come: età, sesso, pratica sportiva ecc."

In SAGE ciò si esprimerà come:

RULE nome-della-regola: "Si determina il fatto che il soggetto sia o meno sovrappeso"

sovrappeso	DEPENDS ON			
età	AFFIRMS	x 1	DENIES	y1
sesso	AFFIRMS	x 2	DENIES	y2
sport	AFFIRMS	x 3	DENIES	y3

*
*
*

PROVIDED (NOT sovrappeso) + 1

La RULE indica che bisogna valutare nell'ordine: età, sesso, la pratica sportiva ecc. (che potranno a loro volta essere OBJECT o ASSERTION domandabili all'utente o determinabili in base a regole) se l'ASSERTION "sovrappeso" è stata dichiarata incerta (PROVIDED).

La valutazione dei vari termini della regola potrà, in base al loro esito, portare ad affermare che l'utente è sovrappeso con una verosimiglianza di volta in volta incrementata di x1, x2, x3 (AFFIRMS) o che non lo è decrementando la verosimiglianza dei valori y1, y2, y3 (DENIES).

Confidando di essere riusciti a dare un'idea di massima sul modo di agire di una RULE non ci si soffermerà a descrivere i meccanismi effettivi del calcolo della verosimiglianza della ASSERTION "sovrappeso", limitandoci a precisare che questi si basano su opportune interpolazioni numeriche.

5.3 Altre caratteristiche del compilatore

Oltre ai costrutti linguistici brevemente introdotti in precedenza, SAGE fornisce altri strumenti per la definizione della base di conoscenza. Fra questi vogliamo segnalare:

- la possibilità di *richiamare funzioni e procedure definite dall'utente (FUNCTION, PROCEDURE)* con lo scopo di risolvere, per esempio, problemi connessi a comunicazioni esterne, ad accessi diretti verso sensori o strumenti, o problemi che richiedono una computazione particolarmente pesante.

- la possibilità di *porre domande all'utente (QUESTION)* in modo più vario di quanto previsto dall'opzione ASKABLE associata ad ASSERTION ed OBJECT.

- la possibilità di *raggruppare domande (QUESTIONGROUP)* in modo che queste vengano richieste di seguito all'utente indipendentemente dal fatto che nel modello siano state, per vari motivi, definite in ordine sparso.

- la possibilità di *potenziare la semantica delle ACTION* mediante l'introduzione di maccanismi quali:

- FORGET: consente di tralasciare la valutazione degli elementi indicati di seguito. (azione opposta a CONSIDER)

- STOP: pone fine all'indagine.

- RESTART: l'intera area viene rivalutata dopo che tutti i risultati raggiunti in precedenza sono stati annullati.

- REASK: consente di riporre alcune QUESTION se il procedere dell'analisi lo rende necessario.

- LATER: consente di rinviare alcune azioni come CONSIDER, ADVISE, ecc. alla fine della valutazione di un'intera area, quando, cioè, non vi sarebbe più nulla da investigare.

5.4 Il supporto per la consultazione del modello.

Il secondo componente di SAGE è l'Executive che ha lo scopo primario di consentire l'accesso al modello. Varie funzioni vengono messe a disposizione dell'utente per consentire, oltre alla normale consultazione del modello una volta che questo è sufficientemente collaudato, una più rapida messa a punto ed un controllo preciso del suo funzionamento.

Nel seguito si elencheranno alcune di queste funzioni.

- WHY
  Questo comando permette di richiamare, per i vari elementi del modello o loro aggregati, il testo esplicativo definito dall'utente nelle varie "string" allo scopo di avere chiarimenti sulle domande che il sistema pone. È possibile precisare anche il livello di completezza della spiegazione richiesta dichiarando il numero di passi del ragionamento che hanno condotto a formulare la domanda e che si vogliono vedere. (es. WHY LEVEL 3 permetterà di vedere gli ultimi tre passi eseguiti prima che la domanda venisse posta).

- LIST
  Consente di avere visione degli elementi del modello. È possibile precisare in modo esauriente gli elementi che si vogliono vedere. Ad esempio, per avere visione di tutte le regole che sono state definite per una data area si imposterà il seguente comando: "LIST RULES IN nome-dell'area". O, ancora, per vedere tutte le ASSERTION, gli OBJECT e le aree che sono già stati completamente valutati si imposterà: "LIST DONE GOALS".

- FACTORS
  Dato un certo elemento del modello, questo comando ha l'effetto di fornire una lista di tutti i fattori che possono influenzare l'elemento stesso ed è in grado di indicare, fra questi, quelli che sono già stati valutati a quel momento.
  Per esempio data l'ASSERTION "sovrappeso", la richiesta "FACTORS sovrappeso" produrrà una lista di tutte le RULE ed ACTION in cui compare "sovrappeso".

- EVALUATE
  Consente di portare a termine una sessione di consultazione senza che l'utente si veda porre ulteriori domande. Il sistema si avvarrà delle domande già poste e dei valori di default impostati nel modello.

- TRACE
  Consente all'utente di avere quattro diversi livelli di visione di quanto sta avvenendo nel modello ed è particolarmente utile nella fase di messa a punto e testing.
  I livelli consentono di vedere dalla semplice informazione generata da domande e da ADVISE di ACTION (livello 0) all'elenco completo di tutti i nodi incontrati durante la consultazione con l'indicazione del loro stato (valutato, non valutato, sconosciuto) e del loro valore (se valutato).

- HISTOGRAM
  Consente di avere una presentazione grafica, sotto forma di diagramma, delle variazioni di valore di un dato elemento durante la valutazione del modello. Anche in questo caso, il comando può essere definito con diversi livelli di potenza.

Ribadiamo che, quello fornito, è un elenco di comandi dell'Executive puramente indicativo.

5.5 Conclusioni

Come esposto in precedenza, non è possibile al momento attuale valutare l'efficienza d'impiego di SAGE nella realizzazione di expert systems sulla base di una qualche esperienza applicativa. Si vogliono comunque richiamare quelle caratteristiche che sembrano renderlo particolarmente appetibile come strumento per realizzare, in particolare, sistemi di consultazione per un supporto alle decisioni.

- È uno "Skeletal Expert" appositamente progettato per essere commercializzato e, come tale, può confortare il suo possibile utilizzatore per quanto riguarda problemi di assistenza tecnica, corsi di addestramento, continuo miglioramento del prodotto. A questo proposito, è annunciata, per la seconda metà del 1984, una nuova versione (SAGE PLUS) notevelmente potenziata rispetto a quella qui considerata.

- Fornisce meccanismi per stimare la verosimiglianza di affermazioni, per esplorare alternative, per se-

guire linee di ragionamento in funzione delle risposte d'utente.

- Accetta ed è in grado di usare sia affermazioni incerte che dati incompleti.
- In un qualsiasi momento è in grado di spiegare esaurientemente il proprio comportamento e di motivare le proprie conclusioni.
- Agisce in costante interazione con l'utente essendo anche in grado di ridurgli il carico di lavoro mediante opportuni raggruppamenti logici delle domande che deve porre.

6. BIBLIOGRAFIA

[1] W.Van Melle et Al., THE EMYCIN MANUAL, Rep. No. STAN-cs-81-885, Standford Univ., Oct. 81

[2] E.H. Shortliffe, COMPUTER-BASED MEDICAL CONSULTATION: MYCIN, American Elsevier N.Y., 1976

[3] Kunz et Al., A PHYSIOLOGICAL RULE-BASED SYSTEM FOR INTERPRETATION PULMUNARY FUNCTION TEST RESULTS, Report HHP, Computer Science Dept., Stanford Univ., 1978

[4] Bennet et Al., SACON: A KNOWLEDGE-BASED CONSULTANT FOR STRUCTURAL ANASISYS, Proc. IJCAI 1979

[5] R.O. Duda, THE PROSPECTOR SYSTEM FOR MINERAL EXPLORATION, Final report, SRI project, 1972

[6] S.H. Weiss, C.A. Kulikowsky, S. Amarel, A MODEL-BASED METHOD FOR COMPUTER AIDED MEDICAL DECISION-MAKING, Artificial Intelligence 11, 1978

[7] S.H. Weiss, C.A. Kulikowsky, EXPERT: A SYSTEM FOR DEVELOPING CONSULTATION MODELS, Proc. IJCAI, 1979

[8] S.H. Weiss et Al., A GUIDE TO THE USE OF THE EXPERT CONSULTANT SYSTEM, Dept. of Computer Science, Rutgers Univ., 1983

[9] SAGE USER MANUAL, SPL Ltd., June 1982

[10] Forgy C.L., OPS5 USER'S MANUAL, CMU - Dept. of Computer Science, 1981

[11] C.L. Forgy, J. McDermott, OPS, A DOMAIN INDEPENDENT PRODUCTION SYSTEM, Proc. IJCAI 1977

[12] J. McDermott, BUILDING EXPERT SYSTEMS, CMU - Dept. of Computer Science, Apr. 1983

[13] S.H. Weiss et Al., BUILDING EXPERT SYSTEMS FOR CONTROLLING COMPLEX PROGRAMS, Proc. AAAI, 1982

[14] S.H. Weiss et Al., DEVELOPING MICROPROCESSOR-BASED EXPERT MODELS FOR INSTRUMENT INTERPRETATION, Proc. IJCAI, 1981

TANGIE. A STORY GENERATING PROGRAM

Graziella Tonfoni (°)

RIASSUNTO

TANGIE è un programma per la generazione di testi narrativi basata su un serie di unità narrative di base; TANGIE può essere definito come un programma "intelligente" perchè è in grado di produrre storie coerenti o di evidenziare incoerenze presenti in una data storia.
Le Unità Narrative rappresentano una serie di concetti e sono basate su criteri di compressione concettuale dei testi narrativi. TANGIE è utilizzabile come strumento per la produzione di sintesi e riassunti.

ABSTRACT

TANGIE is a program for generating stories by using a set of Plot Units (PU). TANGIE can be defined as an "intelligent" program because it creates stories which are coherent. Plot Units represent concepts; the stories which are produced by TANGIE are based on a conceptual understanding of narrative texts. TANGIE works also in summarizing stories by using a PU identification system.

1. INTRODUCTION

What we are presenting here is a model for story generation.

Such model can be used for a whole set of applications; instead of mapping a combinatorial syntax of Plot Units we can think of using a syntax for combining data sets which are logically/causally related such that if we know about some of them we'll be able to assume that some other ones will follow and some other ones are likely to come first.

A very plausible application can be seen in data-manegement systems. Such applications will give a significantly intelligent interface.

Last developments in Natural Language studies have also shown the relevance of finding a model which can be applied for practical purposes especially in relation to Expert Systems in different areas of expert knowledge.

Aim of such research is the organization of expert knowledge and the organization of expert use of expert knowledge by not necessarily expert users.

What can be applied and transferred from a story-model to a management and information retrieval system is the general structure of the model.

2. WHAT IS A STORY GENERATOR?

A Story Generator is a program for generating narrative texts.

A Story Generator helps in organizing information about facts and events by putting them in to a right order. We'll present here TANGIE, a Story Generator which creates stories on an "intelligent" level which means on a conceptual levels.

TANGIE won't just juxtappose a set of sentences in a sequental order; it will rather link together events and facts which are logically or temporally related.

A lot of research on Story-Understanding has been going on in these last years in quite a few related research fields such as Computer Science, Artificial Intelligence and Cognitive Science.

R. Shank formulated a complex theory on Text-Understanding based on meaning analysis of sentences and texts. Such theory is very well known as Conceptual Dependency Theory (CDT). Such theory will allow us to interpret texts on a conceptual level.

Further research developments within such framework have reached an even higher level of description of story-events.

Part of this work of abstraction in concepts recogni-

This research has been done at Bolt Beranek and Newman, Cambridge, Mass, USA. The paper circulates as BBN Technical Report, December 1983.

(°) Bolt Beranek and Newman - Cambridge, MA, USA - Università di Bologna.

tion in narrative texts has been done by W. Lehnert. Her work has been particularly aimed toward concepts identification, concepts representation and concepts organization within narrative texts.

What Lehnert has been using as base-entity of her analysis is a set of Plot Units (simple Plot Units and complex Plot Units, which are basically an aggregation of simple Plot Units).

Lehnert states that Plot Units are usually identifiable while reading stories because they actually represent a way of interpreting sets of events and facts.

TANGIE, the Story Generator which is presented in this paper, works by using Plot Units and a combinatorial syntax of Plot Units, in order to organize a whole set of informations which are related to a specific event or a set of specific events.

Let's now have a look at TANGIE in a more detailed way.

3. WHAT IS A STORY?

A Story is a Narrative Text, which means that it is a set of Events caused by Mental States, and of Mental States caused by Events. (See Lehnert 1980). Mental States are the psycological situations which cause some reactions or justifie some decision-making like in:

John was unhappy so that he decided to have a vacation.

Events are the situations and actions wich take place in a story; they can also cause particular Mental States like in:

John lost his wallet so that he got really depressed.

Mental States (MS) and Events (E) are casually and temporally related. We'll focus our attention on causally related Mental States and Events. Causally related MS and E constitute a set of Simple Plot Units (PU).

A set of casually related Simple PU constitues a complex PU. Plot Units can be represented by one simple or a complex sentence or by more than one sentence as in the following example:

a) John changed his mind.

a') John decided that that trade was too risky, called up Mike and cancelled his meeting.

Both a) and a') reference the "CHANGE OF MIND" Simple Plot Unit, although they differ in their complexity.

In a) it is actually quite easy to identify the PU; in a') we have to think more in order to be able to attribute the correct value expressed by the sentence. In other words, the PU value is explicitly given in a) but has to be inferred in a'). a') could be considered as a "SOLUTION" PU, given the verb *(decides)* which comes first, although the global meaning of the sentence is related to a "CHANGE OF MIND" PU.

In order words, there is no one to one correspondence between a specific verb and a concept expressed by a PU except in a very few cases such as the following:

to want to be willing to to desire	MOTIVATION
to succed	SUCCESS
to fail to miss to need	FAILURE
to change one's own mind	CHANGE of MIND
to lose	LOSS
to keep doing, being, feeling	PERSEVERANCE

to solve	RESOLUTION
to be enabled to enable	ENABLEMENT

The combinational Syntax of PU we will present is based on the expectations a reader tends to have while reading a narrative Text (Story) which is missing some parts of the plot. The process of replacing the missing parts can be forward or backward directed (bidirectional); in other words, we can have two cases:

1) given A+B+C... replace missing D where A+B+C are causally related PU;

2) given B+C+D... replace missing A where B+C+D are causally related PU.

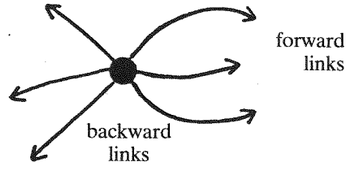
Case 1) is based on a forward linking procedure and is also called "guess what might follow procedure".

Case 2) is based on a backward linking procedure and is also called "guess what might preceded procedure".

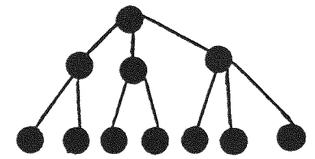
In generating a narrative Text, which means a Story, we won't constrain the complexity and lenght of the sentences in any way, unless such complexity will create ambiguity or difficulties in the general process of understanding of the Story.

4. HOW DOES TANGIE WORK?

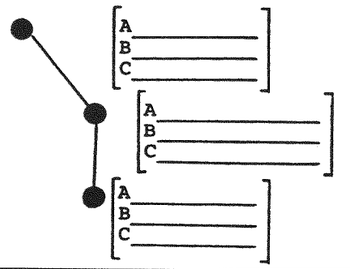
1. TANGIE has a combinatorial Syntax of PU



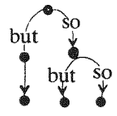
2. Given an initial PU, TANGIE produces a tree of PU, which means a set of possible stories.



3. TANGIE can have a whole set of sentences which correspond to each PU-node, which means to a specific concept.



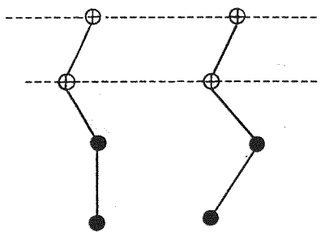
4. TANGIE has a set of possible links (forward and backward) which are represented by "so" and "but" connectors. "so" represents the same goal orientation of two causally related PUs, "but" representes a contrasting goals-orientation of two causally related PUs.



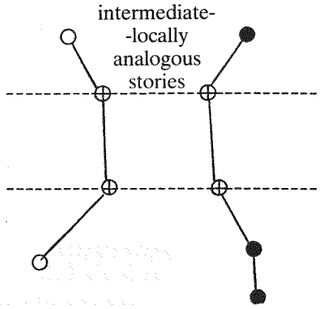
5. TANGIE can produce coherent stories by matching those sentences (which are in each PU-node) and linking them to other ones which are in other PU-nodes by referring to ECSPU

6. TANGIE produces/regonizes locally/globally analogous stories by using ESPU.

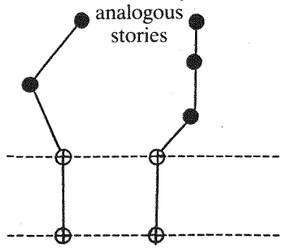
initial-locally analogous stories



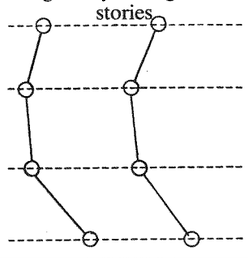
intermediate--locally analogous stories



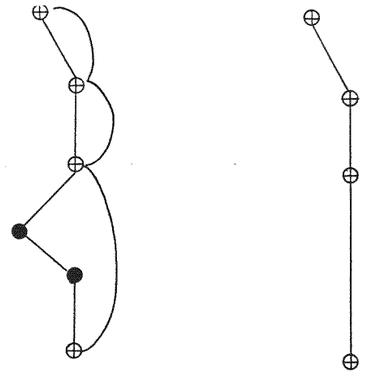
final-locally analogous stories



globally-analogous stories



7. TANGIE summarizes a story once given a premise (which means a PU configuration) by referring to ESPU



5. WHAT DOES TANGIE DO?

The program TANGIE has an elementary combinatorial syntax of PU. TANGIE knows how to link forward and backward; for example, TANGIE knows that PU "MISLEADING SUCCESS" always links backward with a SUCCESS PU and that a PROBLEM PU might be followed by a SOLUTION PU or a SUCCESS PU or a LOSS PU.

This Syntax can be definded as an Elementary Combinatorial Syntax for PU (ECSPU). ECSPU is the result of a wide-spread testing on Story Understanding (see Tonfoni 1983). Some PU-links which have been analyzed are evident because the reader tends to associate and aggregate some PU rather than other ones. For example, nobody would consider a sequence like the following one as totally adequate and acceptable:

John wanted to buy a bike but didn't have money enough. John was happy.

FAILURE - PROBLEM - SUCCESS

This Text has to be considered abiguous and unacceptable because the third assertion *(John was happy* - SUCCESS) is contradictory with the first ones, and the second *(John wanted to buy a bike but didn't have enough money* - FAILURE - PROBLEM) unless we add the sentence *"John won at a lottery"* in between (FAILURE PROBLEM and SUCCESS). As a matter of fact *"John won at a lottery"* corresponds to a RESOLUTION PU which will link backward to the PROBLEM PU and forward to the SUCCESS PU.

TANGIE has a set of sentences which represent the different PU; this way TANGIE can generate a Story by using ECSPU and by ordering and linking sentences in a causal order.

By matching sentences and syntactically ordered PU, TANGIE can produce all possible combinations, which means it can generate all possible Stories by using all combinatorial possibilities according to ECSPU.

TANGIE can also produce a more constrained set of possible Stories, once given a Premise, an Initial PU and a Final PU. TANGIE can also generate a Summary of a given Story by recognizing and linking those PU which are part of the PU Configuration given by the Premise.

TANGIE can also generate a set of partially or globally analogous Stories and recognize partially or globally analogous Stories, once given two Texts which have the same PU structure (globally analogous Stories) or a partially analogous PU structures (locally analogous Stories). TANGIE can also generate one or more possible conclusions for a given Story.

TANGIE can also replace one or more missing PU in a given Story by referring to ECSPU. This is the actual ECSPU scheme which has been currently used in story generation.

List of Currently Used Plot Units and Abbreviations

MOTIVATION	M
FAILURE	F
SUCCESS	S
PERSERVERANCE	PS
LOSS	L
ENABLEMENT	EN
NEGATIVE TRADE OFF	NTO
HIDDEN BLESSING	HB
MIXED BLESSING	MB
POSITIVE TRADE OFF	PTO
PROBLEM	PR
RESOLUTION	RES
CHANGE OF MIND	COM
COMPLEX NEGATIVE EVENT	CNE
COMPLEX POSITIVE EVENT	CPE

1. Phychological testing has been done by G. Tonfoni with a pretty wide series of subjects. See G. Tonfoni - B. Bruce, GUIDELINES TO CHECK COHERENCE IN EXPOSITORY TEXTS, BBN Technical Report, Bolt, Beranek and Newman, Cambridge, Mass 1983.
For a better understanding of the notion of premise, see G. Tonfoni - R. Doyle (1983)

6. PU AND PUA: A COMBINATORIAL SYNTAX. PLAUSIBLE AND POSSIBLE LINKS.

Each simple PU can be combined with another PU or possible set of PU; we can also say that a simple PU can produce differing PU-Aggregations by linking forward and backward by using the ECSPU.

The linking process is based on causal relations.
We can represent a narrative Text (a Story) as a tree which represents the combinatorial possibilities of a set of PU. The forward links (FL) are linguistically represented in the Story by the use of the following set of connectors:

(I) **and-so-so-that-0:** when PU which are causally related and linked together have the same goal-orientation and there is no conflict between the events which have been presented.

Example: *John wanted to buy a ticket to fly to Washington.*
So (and, 0) he called the travel agency and made a reservation for the 8 a.m. shuttle.

(II) **but-0** when 2 PU which are causally related and linked together present events which have different or conflicting goal-orientation.

Example: *Paul decided to buy a new apartment but (0) he found out that he didn't have money enough.*

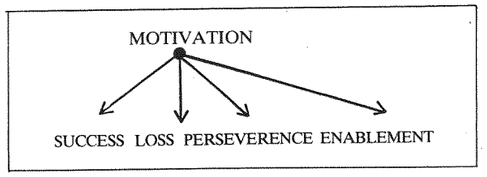
The backward links (BL) are linguistically presented in the Story by the use of the following connectors:

(III) **because-0** in BL no distinction has been drawn between same or contrasting; goal-orientation; "because" is used when the latter of two PU which are causally related and linked together is a direct consequence of the former.

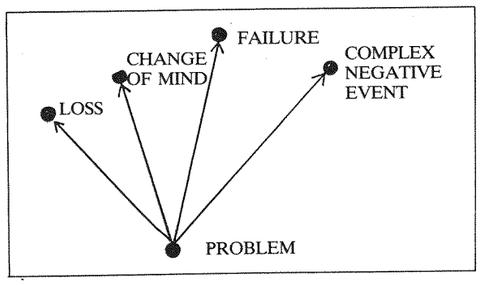
Example: *John called the travel agency and made a reservation for the 8 A.M. shuttle because he wanted to by a ticket to fly to Washington.*

FL can also be defined as Possibility Links (PS-L) because each PU can link forward with a set of possible PU, which are given by the general ECSPU. This procedure has been defined as the "guess what might follow" procedure.

Example:



BL can also be defined as Plausibility-Links (PL-L) because specific PU, can link backward with the precedent one within a PU-Configuration based on ECSPU.



Any particular Story which can be generated by TANGIE is the result of a set of PU which are linked together by using the correct ECSPU.

To be acceptable, a Story, which means a set of logically related PU representing a PU-Configuration, must conform to the following principles:

(a)**Consistence Principle:** A PU-Configuration is consistent when it respects the ECSPU.

(b) **Persistence Principle:** Every PU-node of a PU-Configuration of a Story-tree can contain different sentences which represent the same PU (which means the same concept): it follows that we can have parallel chains of events which represent the same PU-Configuration. A Story is persistent when it has logical-temporal-topic-continuity. A Story is persistent only if it is also consistent. A Story is acceptable only when it is both consistent and persistent.

(c) **Coherence Principle:** a PU-Configuration can be defined as coherent when it is consistent, which means that is respects the ECSPU and a set of appropriate connectors, and is persistent too.

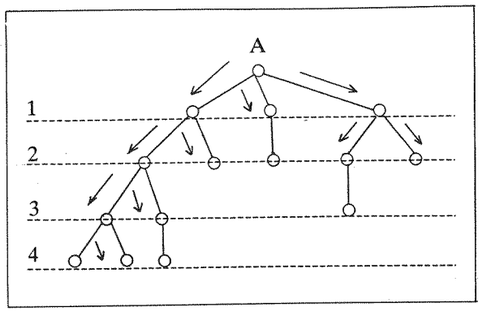
A Story can be more or less complex: this means that we can have a Story which represents a complex chain of PU or a Story which represents a simple chain of PU.

The number of possible PU depends on the number of levels a Story has.

Every PU which is linked forward or backward with another one represents one level of complexity of the Story.

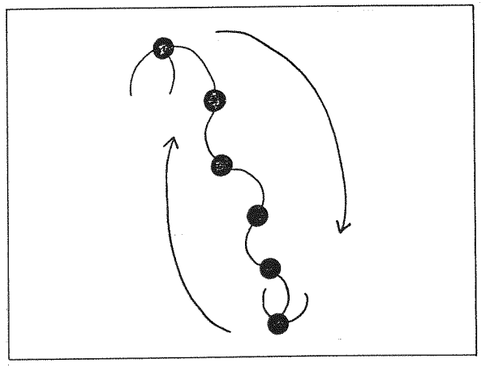
By using ECSPU, we can always create a set of possible Stories, once given an initial PU (by forward linking (FL)) or given a final PU (by backward linking (BL)).

In such a case we don't have any constraints in producing all possible Stories we want. We can represent FL-Text-Generation in the following way:



Where an initial PU (A) has 3 first level possible FL; 5 second level possible FL, 3 third level possible FL, 3 fourth level FL; with respective differing conclusions.

We can also constrain the Text-Generation by previously giving some specific constraints. Let's consider the following case: Given ad initial PU, and a premise and a final PU create a correct Story. (Fig. 1, following page)



In this case we will constrain the generation of Stories only to those which are compatible with respect to a given Initial PU and the Premise PU-Configuration.

We can constrain the possible set of Stories which could be generated by specifying the number of levels (nodes of causally related PU) that a specific Story must have.

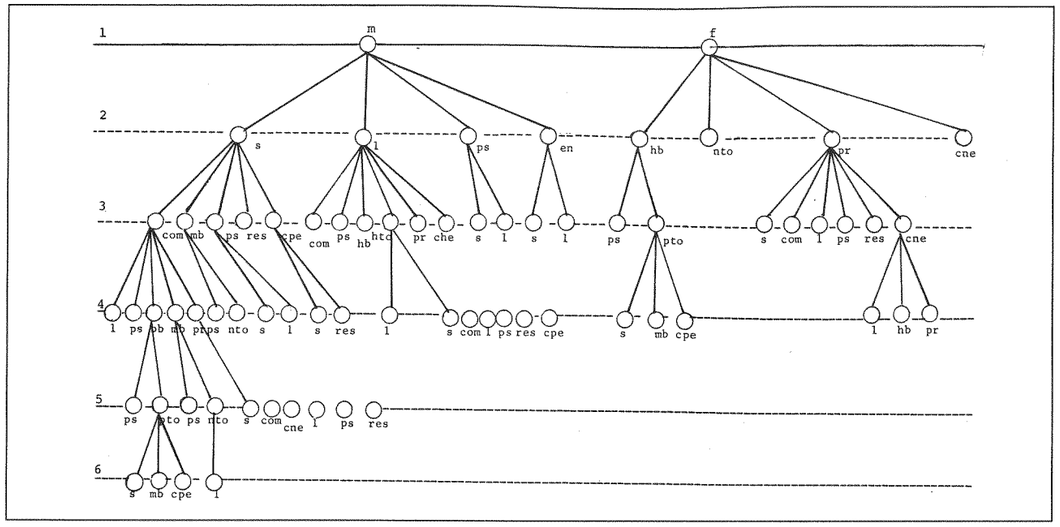
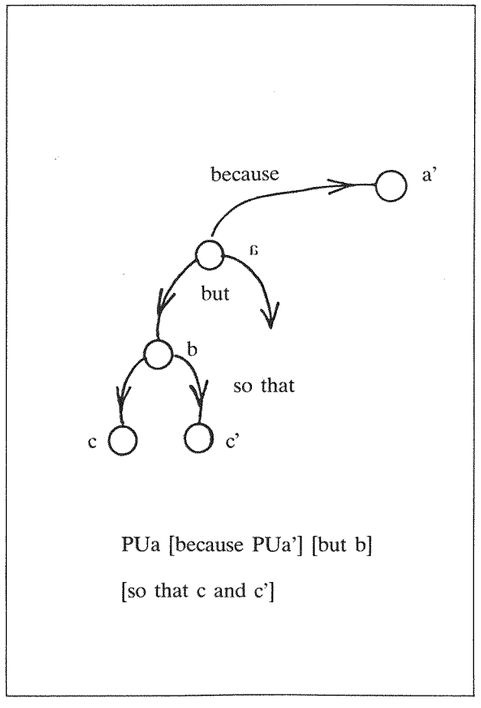


Fig. 1 ECSPU = Elementary Combinatorial Syntax of Plot Units.

We can also add more constraints like in the following case: given an Initial PU and a set of specific links, we want the Story to have the same structure represented in the following configuration.



7. WHAT IS STORY PLANNING?

In order to be well-produced and correctly understood, a Story must have a well formed and coherent Story Internal Planning (SIP).

SIP consists of an aggregation of PU which respects the Consistence Principle and the Persistence Principle as well and is compatible with the PU-Configuration given by the Premise.

In other words, if we have a Story which begins with PUa and ends with PUf, we have to be able to recogize all possible ways to get from PUa to PUf be referring to ECSPU.

Let's consider the following example:

Story A [PUa – – – – – PUf]

How to get from PUa to PUf? Look at ECSPU for Story A, given PUa and Premise X.

SIP may be (a) ⇒ 4 levels of complexity
(b) ⇒ 3 levels of complexity
(c) ⇒ 6 levels of complexity

where: SIP (a) ⇒ PU [1,2,3,4]
SIP (b) ⇒ PU [1,2,3]
SIP (c) ⇒ PU [1,2,3,4,5,6]

Any SIP has to be consistent, persistent and coherent with respect to its final goal which is represented by the Final PU.

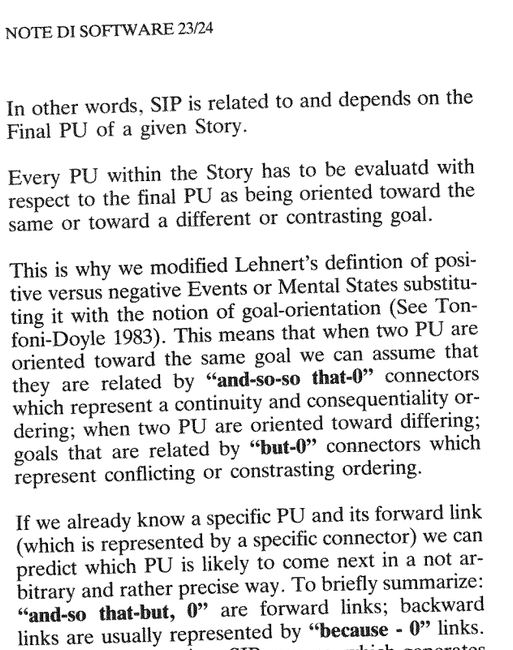
In other words, SIP is related to and depends on the Final PU of a given Story.

Every PU within the Story has to be evaluatd with respect to the final PU as being oriented toward the same or toward a different or contrasting goal.

This is why we modified Lehnert's defintion of positive versus negative Events or Mental States substituting it with the notion of goal-orientation (See Tonfoni-Doyle 1983). This means that when two PU are oriented toward the same goal we can assume that they are related by **"and-so-so that-0"** connectors which represent a continuity and consequentiality ordering; when two PU are oriented toward differing; goals that are related by **"but-0"** connectors which represent conflicting or constrasting ordering.

If we already know a specific PU and its forward link (which is represented by a specific connector) we can predict which PU is likely to come next in a not arbitrary and rather precise way. To briefly summarize: **"and-so that-but, 0"** are forward links; backward links are usually represented by **"because - 0"** links. In order to constrain a SIP-process, which generates a particular PU-Configuration, we can also create a previous Text - structure which presents intermediate level PU.

Such PU are called Bridging PU (BPU) and are particularly important to replace the missing PU during a guessing procedure. A Story can be represented in the following way:



A Story can have different protagonists and can be read by using different points of view; this means that the same sentence can be evaluated in different ways depending on the specific perspective which has been used.

In order to allow a multiple interpretation of the same Story according to different perspectives we'll introduce "demons" in to the SIP-Process. A demon establishes a relation between two possible interpretations of the same sentence.

Let's consider a Story about a competition-situation between Protagonist A and Protagonist B. It is evident that an Event which can be evaluated as being positive for protagonist A will have to be interpreted as negative for protagonist B; a specific demon will be able to give the different protagonists perspectives by establishing specific relations between the contrasting evaluations of the same sentence in the following way:

any time evaluate — sequence A is Prot X's PU-SUCCESS
sequence A as Prot Y's PU-LOSS

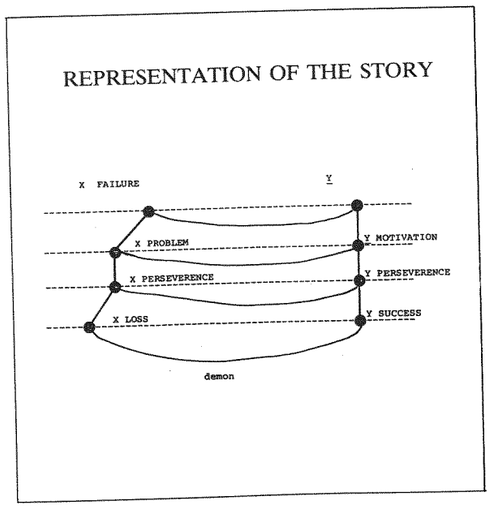
Just to give a simple example let's have a look at the following Story:

1 *John wanted to buy a bike* - J's MOTIVATION

2 *John's mother didn't want him to buy a bike* - J's mother's MOTIVATION - J's PROBLEM

3 *John's mother decided to be severe* - J's mother's PERSEVERENCE - J's PERSEVERENCE

4 *John's mother refused to lend him money* - J's mother's SUCCESS J's LOSS



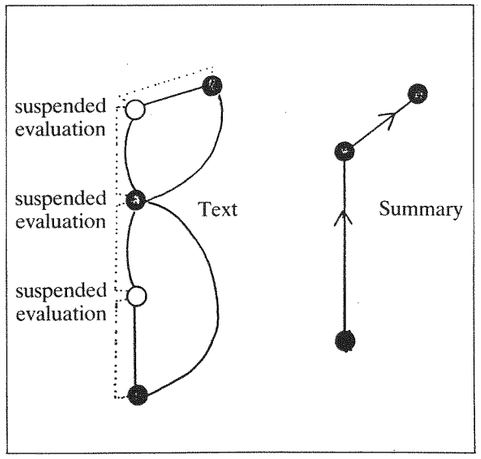
8. SUMMARIZATION PROCEDURES

A summary of a Story consists of an Aggregation of those PU which are relevant both to the Premise and the Conclusion. In other words, a summary is based on a selection of that information, which is in the Text, which has to be kept, in order to allow a reader to really understand a Story.

A Summary, (given a Premise a Conclusion), consists of a set of intermediate PU which can be found through a BL-procedure, starting from a given FPU, TANGIE looks for the next PU, which is both consistent and persistent with FPU before actually linking it with FPU.

TANGIE looks for the second next, if the second next PU is still both consistent and persistent, it moves up and looks for the third next until it finds a PU which is still persistent but no longer consistent; in such case TANGIE establishes a link between FPU and PU.

The search process has intermediate levels called "Suspended Evaluation Steps" which happen to be before an appropriate PU is found. A summarization-Process can be represented in the following way:



A Summary of a Story can also be defined as the shortest possible path between the IPU and FPU. A summary is corrected when it is Consistent, Persistent and Coherent.

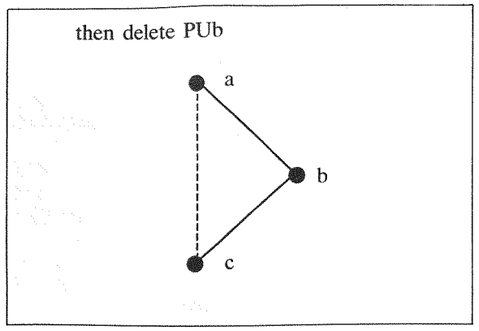
The process of eliminating one or more intermediate level PU is defined as "Jumping Process".

In a PU sequence where

PUa linked to PUb

PUa linked to PUc

if PU a consistent with PUc
and PU a persistent with PUc



A Summary of a Story has to be analogous to the Base-Story which means that it has to keep the same logically ordered set of PU.

9. WHAT IS ANALOGY? WHEN ARE STORIES ANALOGOUS?

Analogy between or among Narrative Texts implies a relation between a Story (the Base Story) and one or more other Stories (the Target Story).

Analogy is not a well-defined and homogeneous concept; there are different kinds and levels of analogy.

Two or more Stories can be defined as globally analogous (gl-a) when they keep the same PU-Configuration; they can be defined as locally analogous (lo-a) if they only partially keep the same PU-Configuration.

Two or more Stories can have the same PU-Configuration but differing linguistic expressions (sentences) to represent the same PU, which means the same concept; such case can be defined as a result of the Different Expressions-Same Nodes Process (DESN); Stories which have the same PU-Configuration and the same linguistic expressions which represent the same PU set are identical; such case can be defined - as a result of the Same Expressions Same Nodes Process (SESN).

We have DESN and SESN both on a global level within the same Story ECSPU.

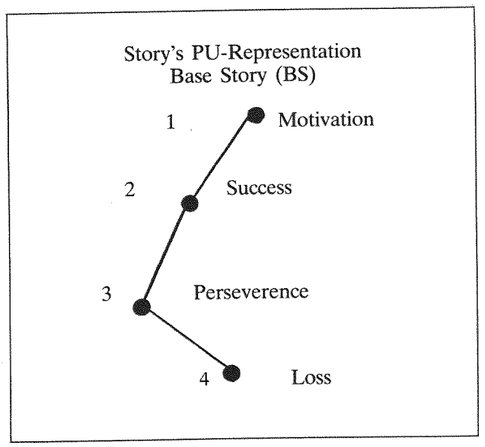
Let's consider the following example:

PU:FAILURE *John needed a bike*
John wanted a bike [DESN]
Paul needed a bike [SESN]

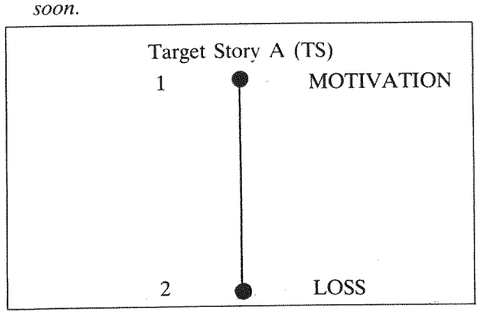
Analogy levels within the same Story plot can be defined as Intra Plot Analogy (Intra P-A); Analogy le-

vels between different Stories can be defined as Inter Plot Analogy (Inter P-A). Intra Plot Analogy and Inter Plot Analogy are based on conceptual organizations.

To give an example of different levels of analogy, lets consider the following set of Stories:



1. *John needed a bike to reach his office*
2. *John borrowed Mike's bike and could reach his office*
3. *John kept biking to his office with Mike's bike.*
4. *John couldn't bike very fast and got tired very soon.*



1. *John needed a bike to reach his office.*
4. *John couldn't bike very fast and got tired very soon.*

TS is gl-a to BS
TS is SESN to BS

TS is a summary of BS

Analogy is too complex to be defined satisfactorily by only one term; this is why we introduced categorizations and distinctions such as globally analogous versus locally analogous and DESN versus SESN.

Analogous Stories generation is the most developed level of Text generation TANGIE can handle.

This last part of the program which is still "work in progress" is based on the same ECSPU which is used for Story-generation.

Psychological testing has shown that different levels of analogy directly influence and determine the way people understand, summarize and memorize Stories and are able to answer questions about them.

In other words, a better way of writing a Text does determine a better understanding and memorizing of that Text.

9. REFERENCES

[1] Black, J.B. and Wilensky, R., AN EVALUATION OF STORY GRAMMARS, Cognitive Science 3, no. 3,213-230, 1979

[2] Carbonell, J.G., INSIDE COMPUTER UNDERSTANDING, Lawrence-Erlbaum Associates Publishers, Hillsdale, New Jersey, 1979

[3] Carbonell, J.G., METALANGUAGE UTTERANCES IN PURPOSIVE DISCOURSE, Carnegie-Mellon C.S. Memo, 82-125, 1982

[4] Dyer, M., Lehnert, W., MEMORY ORGANIZATION AND SEARCH PROCESSES, Dept. of Comp. Sci., Research Report 175, Yale Univ., 1980

[5] Dyer, M.G., IN DEPTH UNDERSTANDING, Research Report 219, Dept. of Computer Science, Yale Univ., New Haven, CT., 1982

[6] Gentner, D., THE STRUCTURE OF ANALOGICAL MODELS IN SCIENCE, BBN Report 4451, Cambridge, 1982

[7] Gentner, D., A STRUCTURE-MAPPING APPROACH TO ANALOGY AND METAPHOR, Proceedings of the International Conference on Cybernetics and Society, 75-79, 1980

[8] Lehnert, W.G., PLOT-UNITS AND NARRATIVE SUMMARIZATION, Cognitive Science 2, 293-331, 1981.

[9] Lehnert, W.G., Alker, H.R., Schneider, D.K., THE HEROIC JESUS: THE AFFECTIVE PLOT STRUCTURE OF TOYNBEE'S CHRISTUS PATIENS, draft, MIT Political Science Dept.

[10] Minsky, M., A FRAMEWORK FOR REPRESENTING KNOWLEDGE, in Winston, P.H. (ed), "The Psychology of Computer Vision", New York: McGraw-Hill, 211-277, 1975.

[11] Minsky, M., K-LINES: A THEORY OF MEMORY, Cognitive Science 4, 117-133, 1980

[12] Minsky, M., WHY PEOPLE THINK COMPUTERS CAN'T, AI Magazine, 3-15, 1982

[13] Minsky, M., ON MEANING, draft, MIT AI Laboratory.

[14] Reiser, B.I., Black, I.B., Lehnert, W.G., THEMATIC KNOWLEDGE, STRUCTURES IN THE UNDERSTANDING AND GENERATION OF NARRATIVES, Cognitive Science Technical Report 16, Cognitive Science Program, Yale Univ., New Haven, Ct., 1982

[15] Schank, R.C., INSIDE COMPUTER UNDERSTANDING, Lawrence-Erlbaum Associates, Hillsdale, New Jersey, 1981

[16] Schank, R.C., DYNAMIC MEMORY: A THEORY OF REMINDING AND LEARNING IN COMPUTERS AND PEOPLE, Hillsdale, NJ., Lawrence Erlbaum, 1982

[17] Schank, R.C., REMINDING AND MEMORY ORGANIZATION: AN INTRODUCTION TO MOPs., in W. Lehnert and M. Ringle (Eds.), "Strategies for Natural Language Processing", Hillsdale NJ., Lawrence Erlbaum, 1982 (in press).

[18] Schank, R. and Abelson, R., SCRIPTS, PLANS GOALS AND UNDERSTANDING, Hillsdale, NJ., Lawrence Erlbaum, 1977. The Artificial Intelligence Series.

[19] Schank, R.C., Lebowitz, M., Birnbaum, L., AN INTEGRATED UNDERSTANDER. American Journal of Computational Linguistics,.1980,6 (I)

[20] Schank, R.C. and Riesbeck, C.K. (Eds.), INSIDE COMPUTER UNDERSTANDING, Five Programs Plus Miniatures, Hillsdale, NJ., Lawrence Erlbaum, 1981

[21] Tonfoni, G., CONDITIONS AND CONSTRAINTS ON ANALOGY PROCEDURE APPLICATIONS, Lingua e Stile, to appear, II Mulino, Bologna, 1982.

[22] Tonfoni, G., PARADOXES AND CENSORS, Semiotica, to appear, Indiana Univ., Press, 1983

[23] Tonfoni, G., A FRAME-SYSTEM THEORY OF TEXT, draft, MIT AI Laboratory, 1983

[24] Tonfoni, G., DALLA LINGUISTICA DEL TESTO ALLA TEORIA TESTUALE, Unicopli, Milano, 1983

[25] Tonfoni, G., Doyle, R., UNDERSTANDING TEXT THROUGH SUMMARIZATION AND ANALOGY, A.I. Memo 716, A.I. Lab., M.I.T., Cambridge, MA., 1983

[26] [Winston, P.H., LEARNING AND REASONING BY ANALOGY, CACM, 23, no. 12, 689-703, 1980

[27] Winston, P.H., LEARNING NEW PRINCIPLES FROM PRECEDENTS AND EXAMPLES, MIT AI Laboratory Memo 632, May 1981.

10. ACKNOWLEDGEMENTS

I want thank Hayward Alker, Bertram Bruce, Dedre Gentner, Wendy Lehnert, and Roger Schank for helpful conversations. I want to thank Mary Marcus, Brenda Starr and Mildred Webster for their help in typing the manuscript.

NOTE SULL'USO DI SISTEMI ESPERTI PER LA DIAGNOSTICA DI GUASTI NELLA STRUMENTAZIONE ELETTRONICA

M. Benedetti
Italservice srl - Milano

RIASSUNTO

In questo lavoro vengono presentate alcune prospettive per lo studio di nuove metodologie (Sistemi Esperti) orientate alle applicazioni nel campo della ricerca guasti in apparecchiature elettroniche.

ABSTRACT

In this work we present some points of view about the studies on new methodologies (Expert Systems) for troubleshooting in electronic devices.

1. IL PROBLEMA

La specifica esperienza degli ambienti del "service" (manutenzioni industriali) ha cominciato a muoversi nella direzione dello studio delle possibilità offerte da nuove metodologie per applicazioni nello specifico campo del troubleshooting (ricerca guasti) in apparecchiature elettroniche e sistemi di regolazione e automazione.

I principali problemi della ricerca guasti nella strumentazione elettronica possono riguardare:

- comportamenti dinamici difformi tra componenti vecchi e nuovi:
- guasti intervenuti in seguito a riparazioni (per es. saldature a mano), dato che la riparazione dell'apparecchiatura non garantisce necessariamente il successivo funzionamento;
- progressiva perdita di efficienza anche senza un guasto vero e proprio dell'apparecchiatura (per es. scheda) dovuta all'uso dei componenti;
- inconvenienti dovuti a fenomeni esterni che fanno risentire la loro influenza all'interno.

Per i problemi riguardanti la ricerca guasti in apparecchiature elettroniche e sistemi di regolazione e automazione, sono stati realizzati sistemi ATE (Automatic Test Equipment) che possono essere utilizzati nelle manutenzioni di apparati elettronici riadattando allo scopo il software di cui sono dotati.

Generalmente si tratta di software ATG (Automatic Test Generation), che è in grado di testare schede elettroniche e rilevare la presenza di guasti con una alta probabilità di riuscita. Tale tipo di testing possiede alcuni limiti fondamentali: per esempio, non è in grado di individuare guasti che compaiono ad alte frequenze di lavoro oppure vengono forniti messaggi diagnostici per la cui interpretazione è necessario l'intervento di specialisti. Nel caso in cui una diagnosi venga fornita direttamente dal sistema ATG, essa non è corredata da una documentazione che ne giustifichi la correttezza.

Nel processo di manutenzione (diagnosi - riparazione) la prima è la fase più impegnativa: possiamo qui brevemente elencare i fondamentali problemi che di solito rendono difficile l'operazione di diagnosi:

- alcuni guasti possono essere mascherati da sintomi provenienti da altri guasti;
- alcuni dati sullo stato del sistema possono essere troppo costosi da rilevare o inaccessibili.
- l'attrezzatura per il rilevamento dei dati può, a sua volta, presentare imperfezioni.

I problemi specifici della ricerca guasti in apparecchiature elettroniche e i problemi generali della diagnostica fanno sì che l'intervento degli specialisti rappresenti ancora il tool di maggior efficacia per la loro soluzione: infatti, l'esperienza continua e, in molti casi, l'intuito, sono capacità oggettivamente rilevanti e non riconducibili ad un modello formale o comunque di assoluto determinismo.

Questa serie di motivi, oltre ad altri classici che riguardano il problema della disponibilità, da parte dell'azienda, dell'esperienza del tecnico specialista anche in sua assenza, hanno indotto, in ambito Italservice, allo studio di possibilità di soluzione a tutta questa serie di problemi mediante un Sistema Esperto.

2. FISIONOMIA DI UN SISTEMA ESPERTO PER QUESTI PROBLEMI.

Come per tutti i casi di progettazione di Sistemi

Esperti, la parte più impegnativa consiste nella costruzione della base della conoscenza: quell'insieme, cioè, di nozioni formali e fattuali che strutturano la conoscenza degli esperti riguardo al loro campo d'azione.

Fra tutti i problemi di rappresentazione della conoscenza, quello più complesso è senz'altro quello della rappresentazione dell'esperienza del tecnico, esperienza che, sebbene per buona parte possa essere ridotta ad elementi separati sui quali operare abbastanza agevolmente (per es. quella tratta dai manuali), presenta comunque dei risvolti che derivano dall'intuizione e dall'ordinario ragionamento di senso comune.

Ovviamente, sono questi ultimi tipi di nozione i più difficili da definire in modo rigoroso.

Probabilmente, la maggior parte dei problemi del troubleshooting si possono ricondurre, con un po' di sforzo, a fatti univoci e chiari e quindi a dei formalismi teoricamente soddisfacenti.

Per ovviare ai problemi di costruzione di una base della conoscenza che tenga conto dei fatti sopra citati, abbiamo iniziato lo studio di un linguaggio che utilizza i seguenti criteri strutturali:

AREE: possono essere definite come differenti zone di investigazione all'interno del modello.

AZIONI: ogni area contiene una serie di azioni che, ognuna identificata in base ad un suo specifico contenuto, raccolgono sotto una unica definizione una serie di operazioni da compiere. Tali operazioni possono consistere in domande da porre al consultante o da asserzioni da valutare o, ancora, da regole da scandire.

ASSERZIONI: rappresentano le principali componenti della struttura inferenziale del modello. Identificano le evidenze fornite dall'esterno e ad esse vengono associati pesi o valori di tipo probabilistico dell'evidenza. Tale valore si modifica durante il processo diagnostico.

OGGETTI: sono elementi simili alle asserzioni ma contengono solo valori numerici.

REGOLE: sono costituite da un insieme variabile di asserzioni alle quali viene associata una evidenza con il suo peso.

Le asserzioni componenti le regole sono combinate tra loro mediante connettori di tipo booleano.

A titolo esemplificativo, presentiamo qui un tipo di dialogo ottenibile mediante il linguaggio sopra descritto (fig. 1).

Oltre alle classiche basi delle regole e dei dati abbiamo introdotto una ulteriore base delle regole che racchiude il modello dell'UUT (Unit Under Test). È stato necessario fare questa distinzione per far sì che il sistema disponesse di un modello comparabile a quello rappresentato da uno schema circuitale nelle mani di un tecnico di manutenzione.

Naturalmente, nella struttura esposta le due basi delle regole non sono scindibili fisicamente; lo sono però logicamente mediante l'opportuno uso delle aree di investigazione: mantenendo la medesima base delle regole generali e variando il modello dell'UUT si variano le capacità di investigazione anche all'interno dello stesso dominio.

Un sistema costruito con questi criteri può dunque

```
AREA investigation: "string"

ACTION find-result: "establish a value for the
                     result"

CONSIDER result

ASSERTION result: "final result"

PRIOR 0.0

RULE example-1: "Ex. cumulative rule to esta-
                 blish the result"

result DEPENDS ON

first-evidence AFFIRMS 8.0 DENISE -3.0

second-evidence AFFIRMS 4.0 DENISE -3.0

ASSERTION first-evidence: "the first piece of
                           evidence is true"

PRIOR 0.0 ASKABLE

ASSERTION second-evidence: "the second pie-
                            ce of evidence is
                            true"

PRIOR 0.0 ASKABLE
```

Fig. 1

essere utilizzato per diagnosticare guasti su più apparecchiature fornendo una sola volta le regole di carattere generale e formando via via il modello particolare di ogni apparecchiatura interessata al processo manutentivo.

3. SEDUTE DIAGNOSTICHE.

In linea di massima un tecnico può isolare il componente che presume avere difetti lanciando una serie di test su componenti di basso livello (per es. diodi, resistenze, transistor, etc.). Tale metodo conduce spesso al successo ed infatti è molto usato laddove la densità circuitale sia bassa; al contrario, dove essa è invece elevata, è necessario provedere ad uno "sgrossamento" del problema mediante l'analisi funzionale e solo successivamente passare al test sul componente.

Questo significa che il tecnico prima individua un'aera che appaia funzionalmente difettosa, quindi, all'interno di questa area, opererà una ricerca per componenti discreti. Si assume con questo che ciascun circuito elettronico sia progettato come una struttura "goal oriented" di subcircuiti ai quali sia stato attribuito uno specifico goal. Anche nell'analisi di circuiti di cui non si conosca perfettamente la struttura "goal oriented" si può sempre procedere per riduzioni a substrutture che abbiano significato e alle quali si possa associare un compito inequivocabile.

È facile notare che ci si trova in presenza di un tipo di analisi gerarchico, dove al livello più alto risulteranno essere posti i sottosistemi cruciali per il funzionamento generale (alimentazione, ingresso, etc.) e ai livelli via via più bassi appariranno i componenti discreti.

Tale tipo di organizzazione consente di ottimizzare il lavoro del tecnico permettendogli di porre la sua attenzione, per ogni livello, solo su un numero limitato di componenti, dall'alto verso il basso.

I problemi di testing riguarderanno solo un livello alla volta, influenzando la scelta del test successivo.

Nel dominio applicativo da noi considerato, le scelte di valutazione sono scandite da fatti naturali e da componenti dell'ambiente circostante che non indicano necessariamente delle evidenze assolute: è conveniente quindi arricchire le evidenze con valori probabilistici. Il peso, infatti, visto numericamente come costo del guasto, può determinare la sospensione della ricerca o la convenienza al suo proseguimento: il molti casi la sostituzione di un componente può presentare maggior convenienza economica rispetto alla sua riparazione.

In fig. 2 un esempio di regole che determinano i legami delle conoscenze.

R6	IF	La luce della fotocellula è debole
	THEN	La lampada IBP non rimane accesa
	ACTION	Regolare i potenziometri R4 e R5
R6a	IF	Azionati i potenziometri R4 e R5
	AND	La luminosità aumenta
	THEN	La luce è stata regolata
R6b	IF	Azionati i potenziometri R4 e R5
	AND	La luminosità non aumenta
	THEN	L'alimentazione delle luci è guasta
R7	IF	La luce è stata regolata
	AND	La fotocellula originale è guasta
	THEN	La lampada IBP non rimane accesa
	ACTION	Provare a cambiare la fotocellula
R7a	IF	Provato a sostituire la fotocellula
	AND	La lampada IBP rimane accesa
	THEN	La fotocellula originale è guasta
R8	IF	Il punto di lavoro della fotocellula è shiftato
	THEN	La lampada IBP non rimane accesa
	ACTION	Regolare il potenziometro P9
R8a	IF	La lampada è macchiata
	THEN	Il punto di lavoro della fotocellula è shiftato
R9	IF	Potenziometro P9 regolato
	AND	La lampada IPB non rimane accesa
	THEN	L'amplificatore è guasto
	ACTION	Sostituire la scheda dell'amplificatore

Fig. 2

In una struttura di questo tipo, le procedure di ricerca che determinano quello che è comunemente conosciuto come il "meccanismo inferenziale" di un Sistema Esperto è ancora in fase prototipale. Le scelte implementative comunque già abbozzate riguardano l'uso del metodo di ricerca di tipo "backward chaining".

Si tratta di un metodo largamente usato nelle passate esperienze applicative "storiche" di Sistemi Esperti di diagnostica il quale ha il pregio di assomigliare, in massima parte, al metodo utilizzato dal tecnico nella sua prassi di ricerca guasti.

L'ipotesi di partenza si riferisce all'aspetto più evidente di non funzionalità dell'apparecchiatura in esame, cioè il suo guasto apparente, cioè rispetto ai sintomi più macroscopici.

Il tecnico consultante deve comunicare al sistema quale area prendere in considerazione per prima, ovvero quali ipotesi di guasto provare.

4. CONCLUSIONI

Abbiamo abbozzato alcuni problemi tecnici affrontati nel campo della diagnostica di apparecchiature elettroniche.

Si è poi accennato ad uno studio di fattibilità di un sistema di diagnosi di guasti seguendo alcune metodologie impiegate in analoghi casi di diagnostica mediante Sistemi Esperti.

5. BIBLIOGRAFIA

[1] Barr, Feigenbaum, THE HANDBOOK OF ARTIFICIAL INTELLIGENCE, Kaufmann Inc., Los Altos, California, 1981, voll. 1-2-3.

[2] Winston, LISP, Addison Wesley Publ. Co., Reading Ma., 1981.

[3] Gini, Gini, Morpurgo, A KNOWLEDGE BASED CONSULTATION SYSTEM FOR AUTOMATIC MAINTENANCE AND REPAIR, Advances in CAD/CAM, North Holland Publ. Co., I.F.I.P., 1983.

[4] Cantone et al., MODEL BASED PROBABILISTIC REASONING FOR ELECTRONIC TROUBLESHOOTING, Proceedings IJCAI, Karlsruhe 1983, vol. 1.

[5] Tanaka, REPRESENTATION AND ANALYSIS OF ELECTRICAL CIRCUITS IN A DEDUCTIVE SYSTEM, Proceedings JICAI, Karlsruhe 1983, vol. 1.

[6] MANUALE DI RIPARAZIONE PER DISTILLATORI AUTOMATICI, Herzog, Herzog D 697, Lauda/Baden, W. Germany.

GLI STANDARD GRAFICI

Daniele Marini
Istituto di Cibernetica - Università degli Studi di Milano

RIASSUNTO

In questa nota viene presentata una sintetica rassegna della problematica degli standard grafici. Tale problematica si è imposta negli ultimi anni data l'enorme crescita del mercato di sistemi, periferiche e software. Si illustrano brevemente le proposte CORE e GKS come standard per la realizzazione di pacchetti grafici applicativi, e dello standard IGES per l'intercambio di dati grafici tra differenti sistemi.

ABSTRACT

In this paper we present a synthetic review of the graphics standards. This problem became of extreme interest in recent years due to the development of graphics systems, peripherals and application software. We describe briefly the CORE and GKS proposal for the implementation of graphic basic packages suitable for the implementation of portable application programs and systems. Moreover we give an outline of IGES, wich suggest a standardized approach to the problem of graphic data interchange among different systems.

1. INTRODUZIONE

Lo sviluppo delle applicazioni della computer graphics nei più diversi settori applicativi ha fatto nascere già da alcuni anni la esigenza di risolvere anche in questo settore il problema della portabilità del software e della produttività nello sviluppo di nuove applicazioni.

Questo problema, nel caso della computer graphics, è aggravato dal fatto che la portabilità di un pacchetto applicativo deve venire garantita non soltanto in relazione al linguaggio di programmazione in cui il prodotto è stato sviluppato, e in relazione al sistema operativo ospite. Infatti la varietà delle periferiche che sono disponibili sul mercato e che vengono utilizzate nei settori più diversi richiedono una altissima indipendenza anche dalle periferiche specifiche. Ricordiamo che tali periferiche sono caratterizzate dall'essere, alcune, interattive, altre puramente batch, come ad esempio un terminale videografico a colori, tipico strumento interattivo, e un plotter, la più tradizionale periferica batch utilizzata in computer graphics.

La problematica della portabilità può, in una certa misura, essere esaurita in questa visione; tuttavia già si è posto in più settori il problema del trasferimento non soltanto di programmi applicativi ma anche di elaborati grafici. In ambienti di progettazione industriale, infatti, spesso sono presenti diversi sistemi grafici e di elaborazione. Ad esempio un sistema grafico può essere dedicato alla creazione di un modello grafico di un progetto, mentre un altro sistema di elaborazione può venire utilizzato per eseguire simulazioni numeriche (ad esempio una modellazione ad elementi finiti). In questo caso è di estrema utilità poter trasferire i dati del modello grafico costruito sul sistema di modellazione al sistema di simulazione del modello.

Per alcuni anni un unico produttore di terminali grafici, la Tektronix, ha di fatto imposto uno standard per la stesura di programmi di visualizzazione di dati. Infatti i pacchetti di subroutine che tale società distribuiva, soddisfacevano i minimi requisiti di portabilità tra i più diffusi elaboratori. Per quanto riguarda la creazione di figure su carta, un'altra casa, la Calcomp, ha a sua volta imposto uno standard diverso anche se molto semplice.

Col diffondersi di nuove periferiche, verso la metà degli anni Settanta la problematica della standardizzazione si è imposta con decisione, portando alla costituzione di un gruppo speciale nell'ambito dell'ACM (Graphics Standard Planning Committee) il quale ha proposto una formulazione, denominata CORE, che, con qualche difficoltà, si è lentamente imposta nel campo dei sistemi basati su terminali di tipo "storage" (ad esempio terminali Tektronix della serie 4010) e nei sistemi prevalentemente dedicati ad applicazioni in ingegneria.

Alla fine degli anni settanta in Germania venne costituito un gruppo analogo il quale ha proposto un diverso standard denominato Graphical Kernel System o G.K.S. Quest'ultimo è stato adottato come standard DIN e successivamente la stessa ISO lo ha fatto proprio, relegando di fatto il CORE a un ruolo sempre più marginale.

Per quanto riguarda la standardizzazione della descrizione di un modello grafico, finalizzata allo scambio di dati grafici tra diversi sistemi, è stato proposto lo standard IGES.

Volendo sintetizzare in un diagramma l'evoluzione della problematica degli standard grafici, nella fig. 1.a è evidenziata la situazione precedente ad ogni standardizzazione, ogni programma applicativo doveva venire scritto in funzione della periferica adottata e del sistema ospite, scrivendo inoltre un pacchetto grafico personale. In seguito, nel secondo schema 1.b, i pacchetti grafici di base sono stati standardizzati da differenti case e l'utente doveva solo preoccu-

parsi di adottarne uno e di scrivere il proprio programma applicativo conformemente ad esso. Infine la comparsa degli standard rende disponibile la situazione descritta nel terzo schema 1.c, in cui il programma applicativo è scritto in funzione di un unico pacchetto grafico standard il quale, inoltre, dispone di diversi device drivers che possono essere inglobati nel prodotto in funzione delle periferiche utilizzate.

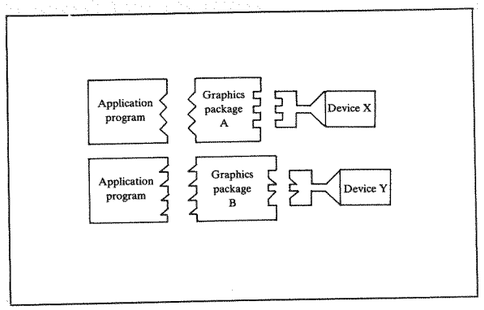


Fig. 1.a

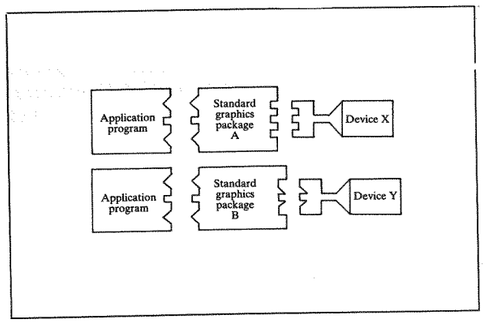


Fig. 1.b

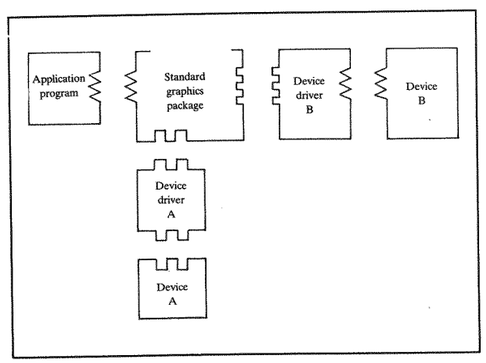


Fig. 1.c - Evoluzione della device dependence

2. STANDARD CORE

Una descrizione articolata dello standard CORE può essere trovata in [1]; qui ci limitiamo a richiamare i concetti essenziali che stanno alla base di questa proposta, evidenziando in seguito le differenze con lo standard GKS.

Lo scopo della proposta CORE del gruppo GSPC dell'ACM consisteva nel formulare i requisiti standard per un pacchetto grafico che fosse nello stesso tempo

– completo
– portabile.

Conseguentemente occorre garantire l'indipendenza da:

– periferiche
– elaboratore ospite
– sistema operativo.

Uno studio accurato di un generico sistema grafico porta a identificare alcune caratteristiche fondamentali:

- Separazione tra funzioni di input e funzioni di output
- Ridurre al minimo le differenze tra realizzazione dell'output su periferiche diverse, in particolare tra videoterminale e plotter
- Ogni sistema grafico si basa sulla distinzione tra coordinate dell'universo in cui il modello in costruzione è definito, e coordinate della periferica su cui il disegno del modello viene visualizzato. Questa distinzione viene evidenziata denominando le prime "WORLD COORDINATES", e le seconde "DEVICE COORDINATES".
- Ogni sistema grafico richiede la gestione di un DISPLAY FILE, ovvero del file che contiene tutte le informazioni necessarie alla visualizzazione della figura su una particolare periferica.
- Il display File può inoltre essere segmentato al fine di consentire una gestione più agile delle modifiche del disegno, dando luogo alla nozione di DISPLAY FILE SEGMENT.
- Infine il processo di visualizzazione di una figura può venire descritto attraverso un pipe-line, in cui a ogni stadio vengono effettuate alcune operazioni fondamentali, quali l'applicazione di trasformazioni geometriche sulla figura, il passaggio da coordinate mondo a coordinate device e il "clipping", ovvero l'esclusione dalla vista delle parti che sono esterne alla finestra di visualizzazione.

Ciò che caraterizza il CORE è la modalità con cui tutti gli aspetti sopra elencati possono venire implementati.

Un ulteriore rilevante aspetto del CORE è l'introduzione della distinzione tra MODELLING SYSTEM e GRAPHIC SYSTEM.
Questa distinzione consente di separare le operazioni di trasformazione che vengono effettuate sull'oggetto che verrà poi rappresentato graficamente, dalle trasformazioni che occorre eseguire per visualizzare l'oggetto su una particolare periferica e dalle trasformazioni che si possono eseguire sulla immagine. Il modelling system supporta il primo tipo di trasformazioni, mentre le seconde sono a carico del graphic system. È di queste seconde che si occupano in generale gli standard e in particolare il CORE, che si configura quindi come pacchetto di subroutines o di procedure richiamabili con sintassi uniforme in programmi applicativi scritti in linguaggi ad alto livello. Con l'utilizzo di un tale pacchetto è dunque possibile svilupare sistemi grafici anche interattivi che permettono il trattamento di modelli tridimensionali anche molto complessi e sofisticati.

2.1 Livelli del Core

Esistendo una molteciplicità di periferiche aventi prerogative a volte molto differenziate, i propositori dello standard hanno provveduto a fornire delle classi di livelli che evitino la nascita e la proliferazione di dialetti di volta in volta comprendenti questa o quella funzionalità.

Le tre classi di livelli, compatibili verso l'alto, sono: uscite, ingressi, dimensioni.

I livelli per le funzioni di uscita del CORE sono:

1 base	primitive, attributi delle primitive, visualizzazione, controllo segmenti temporanei;
2 bufferizzato	segmenti ritenuti, attributi del segmento;
3 dinamico	attributi di trasformazione, attributi di sensibilità.

I livelli per le funzioni di ingresso del CORE sono:

1 nullo	
2 sincrono	inizializzazione della periferica, interazione sincrona, controllo dell'eco:
3 completo	abilitazione e disabilitazione esplicita, controllo code di eventi, campionatura, associazioni.

I livelli per la dimensionalità del CORE sono quelli illustrati nella fig. 2.

1 2D	tutte le funzioni sono nel piano;
2 3D	tutte le funzioni sono nello stesso spazio e sono visualizzati gli oggetti contenuti in un tronco di piramide.

2.2 La pipe-line di visualizzazione

La visualizzazione di una figura avviene secondo la sequenza delineata in fig. 2.
Secondo questa sequenza, ogni volta che si desidera operare una trasformazione che può provocare la comparsa o scomparsa di elementi di figura clippati, ciò dovrà essere svolto sul modello dal programma applicativo, operando sulle coordinate mondo, in quanto l'informazione clippata in coordinate della device, una volta completata la pipe-line, sono perse.

3. STANDARD GKS

Gli elementi che caratterizzano la proposta GKS si possono sintetizzare in una serie di proposte:

- la workstation, ovvero l'insieme di parametri specificamente legati a un particolare set di periferiche di input ed output
- il metafile, ovvero la possibilità di memorizzare un insieme di informazioni grafiche in un file grafico permanente e indipendente dalle workstation.

Per altri aspetti la proposta GKS è ispirata al CORE, salvo, almeno allo stato attuale, l'assenza della gestione delle tre dimensioni e la presenza di primitive per la gestone di terminali raster.

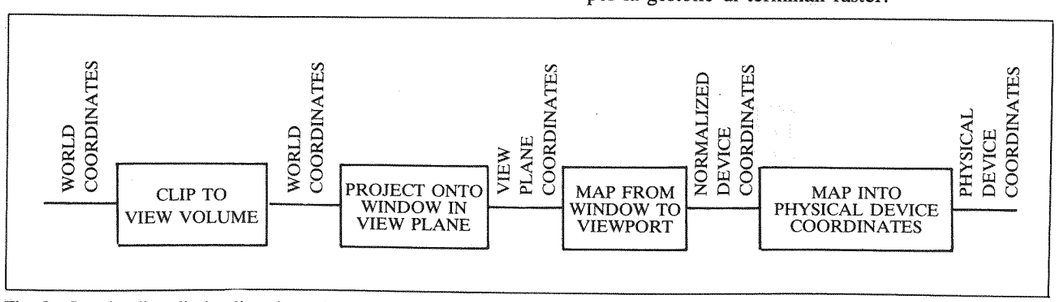


Fig. 2 - La pipe-line di visualizzazione del core

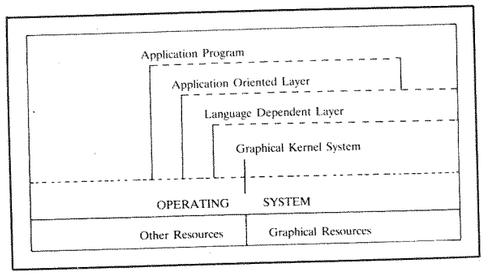


Fig. 3

Un ulteriore elemento di estrema importanza del GKS è la indipendenza dal linguaggio di implementazione adottabile: lo standard consiste infatti solo in un nucleo che per l'implementazione richiede l'ambientazione nel linguaggio adottato.

In fig. 3 è riportato il rapporto tra GKS e gli altri elementi di un sistema grafico.

Dal punto di vista delle primitive il GKS dispone di primitive per la creazione di poligonali, (POLYLINE), di simboli centrati (POLYMARKER), di testi (TEXT), di tracciamento di curve speciali come spline, archi ecc. (DRAW); le primitive orientate a display raster sono FILL AREA, per il riempimento di poligonali con un colore uniforme, e PIXEL ARRAY per visualizzare un array di pixel con colori specifici.

Anche GKS consente segmentazione di immagini; primitive raggruppate in segmenti consentono l'applicazione di operazioni particolari, mentre primitive non segmentate vengono visualizzate su tutte le workstation attive in un particolare momento.

A ogni primitiva è associata una serie di attributi, tra cui la rilevabilità (PICK); gli altri attributi sono distinguibili in statici e dinamici. I primi sono il colore (o numero di penna), il numero di testo (che caratterizza la fonte utilizzata), le dimensioni del testo e del marker. Attributi dinamici sono la visibilità, la reperibiltà, la luminosità, la priorità e le trasformazioni. Inoltre gli attributi dinamici possono essere associati alla workstation attiva.

I sistemi di coordinate gestiti dal GKS sono le coordinate reali (WC), le coordinate normalizzate di device (NDC) e le coordinate del device (DC). Il programma applicativo definisce le primitive di output in WC che attraverso una trasformazione vengono convertite in NDC e successivamente in DC. Il clipping in coordinate DC è automatico, mentre quello in coordinate WC può essere richiesto dall'utente.

Altre primitive del GKS permettono la gestione dell'input.

Si tratta di cinque classi che permettono di restituire i parametri geometrici relativi alla identificazione di una entità grafica o a una azione.

La gestione dell'input può essere effettuata con tecniche di polling o di interrupt, consentendo all'utente di generare code di richieste di input e di gestirle liberamente.

Il GKS viene solitamente descritto sulla base di livelli:

livello 0 capacità minimali di output, nessun input, nessun attributo alle stazioni

livello 1 output completo, controllo della workstation, gestione del metafile

livello 2a comprende anche la segmentazione e la gestione di attributi

livello 2b l'input è semplice e sono disponibili attributi statici associati alle stazioni; inoltre non è disponibile la segmentazione

livello 3 output completo, segmentazione e attributi di segmenti, input parziale, attributi statici di stazione

livello 4 completamento delle capacità del GKS

Il metafile del KGS consiste nella possibilità di memorizzare informazioni grafiche, attuata mediante una interfaccia che consente la memorizzazione di dati grafici sul metafile GKSM. Per omogeneità il GKSM è visto come una workstation.
Dal punto di vista implementativo non è altro che una sequenza di record variabili preceduti da una intestazione; i record sono tipizzati in modo da semplificare la interpretazione del metafile.

4. STANDARD IGES

IGES (Initial Graphics Exchange Specification) consiste in una proposta per realizzare velocemente un insieme di specifiche che permettano lo scambio di dati tra differenti sistemi CAD/CAM commerciali. I modelli sono descritti per linee e superfici e le informazioni vengono estratte dai data-base dei sistemi e riorganizzate in modo da consentire anche l'accesso di programmi applicativi. La motivazione di IGES deriva dalle pesanti limitazioni poste dai sistemi "turnkey", i quali, se da una parte si sono dimostrati potenti per supportare l'attività di modellazione grafica, dall'altra hanno anche mostrato enormi difficoltà sia nell'evoluzione e nell'adattamento che nella possibilità di supportare altre fasi del processo di progettazione, quali ad esempio la simulazione o il calcolo strutturale.

La logica alla base di IGES è quella della costruzione di files strutturati, che contengono tutte le informazioni necessarie alla ricostruzione del modello.

5. CONCLUSIONI

La comparsa negli ultimi anni di sistemi quali PERQ, Apollo, SUN, PS300 ecc. sta modificando la importanza e le caratteristiche degli standard grafici. Questi sistemi infatti possono caratterizzarsi sia come elaboratori stand-alone con prestazioni grafiche evolute, che come workstation in grado di ridurre il volume di elaborazione dei calcolatori ospiti ai quali vengono collegate.

Questi sistem si collocano in una prospettiva che vede il CAD sempre più simile alla automazione di ufficio; in sostanza si identifica la tendenza verso l'ufficio di progettazione assistita dall'elaboratore. In tale senso le workstation evolute si presentano come posti di lavoro dotati di utilities di vario genere, non esclusivamente limitate alla grafica, e sono tese a sfruttare al massimo le prestazioni degli ambienti di rete locale in cui vengono inserite, mettendo a disposizione file servers, word processing, e facilitazioni per lo sviluppo software nei casi in cui ciò è necessario.

Le problematiche che queste caratteristiche pongono alla standardizzazione invitano a rivedere sia il concetto di "device independence" che quello di compatibilità tra diversi data base grafici.

Gli aspetti più rilevanti che comportano una revisione dei concetti base sugli standard sono:

* incorporazione di funzioni grafiche evolute, a livello firmware. Ciò dà luogo a un aumento delle prestazioni, che sgrava l'elaboratore ospite dalla necessità di svolgere alcune funzioni. Quelle più diffuse sui diversi sistemi sono:

  ** tracciamento di primitive grafiche (linee, cerchi, rettangoli, caratteri ecc.),

  ** tracciamento di entità grafiche più evolute (spline, coniche ecc.),

  ** esecuzione di clipping bi o tridimensionale,

  ** gestione del display file in forma gerarchica,

  ** filling di aree con pattern qualsiasi,

  ** operazioni raster che permettono panning, rotazione o pseudo-zoom,

  ** antialiasing per la riduzione della seghetatura delle linee.

In alcuni esempi di superstazioni dedicate alla grafica, quali il PS300 della Evans & Sutherland, si ha a disposizione una configurazione completa, comprendente sistema operativo, ram che può arrivare ad alcuni megabytes, linguaggio interpretativo per la esecuzione di comandi e di istruzioni grafiche evolute, architettura del sistema di visualizzazione di tipo pipe-line. Il panorama che si viene a costituire costringe a esaltare la nozione di workstation proposta dallo standard GKS e a considerare molto criticamente l'approccio del CORE.

Le maggiori riserve che è possibile avanzare sono:

* L'assenza del concetto di workstation ha conseguenze gravi sia dal punto di vista della efficienza (sfruttamento ottimale delle prestazioni del terminale utilizzato), che dal punto di vista della integrità del pacchetto di procedure. Si pensi ad esempio alla disponibilità nella stazione di lavoro di istruzioni, magari firmwarizzate, per eseguire clipping, tracciare archi ecc. In tali casi la soluzione può richiedere la modifica di procedure del pacchetto standard e la rottura della pipe-line di visualizzazione, poichè si può escludere il clipping via software di archi, di stringhe di testo, ecc.

* La struttura della pipe-line di visualizzazione nel CORE è tale per cui ogni segmento viene memorizzato nel display file di coordinate normalizzate dopo che sono state eseguite le procedure di trasformazione e di clipping sulle primitive componenti il segmento stesso. Di diretta conseguenza le trasformazioni di immagine proposte seguono un approccio che presenta inevitabilmente numerosi difetti:

  ** essendo successive alla fase di clipping sono inutilizzabili per fini di animazione della scena a causa della incompletezza delle informazioni su ciò che stava fuori della piramide;

  ** se la trasformazione porta fuori della piramide di visualizzazione qualche vertice sarà necessaio un nuovo clipping;

  ** La computazione delle trasformazioni resta pesante essendo svolta in 3D e quindi utilizzando operazioni su matrici 4x4;

* Un set di operazioni raster locali sono molto più complete ed efficienti; larga parte delle operazioni di editing grafico (anche nella modellazione tridimensionale) sono infatti compiute in due dimensioni; si pensi ad esempio al "dragging" di sagome complesse, alla gestione di window, agli effetti raster per il colore ecc.

* La struttura della pipe-line di visualizzazione e il tipo di dati memorizzati nel display file (coordinate normalizzate su cui viene eseguito il clipping

rispetto alla window), rendono necessaria una completa riesecuzione ogni qualvolta si richieda uno spostamento della window. Ciò comporta un maggiore appesantimento del programma applicativo, il quale deve farsi carico della gestione del file della figura e di tutte le trasformazioni che su questa devono venire eseguite su richiesta dell'utente. Nel caso di terminali a rinfresco (sia a vettori che raster) questo fatto permette di sfruttare solo parzialmente le potenzialità di tali devices in operazioni ad esempio di "dragging". Infatti in tal caso sarebbe utile avere nel display file le coordinate "world" relative, e applicare su di esse trasformazioni locali governate con strumenti di interazione.

* Sorge inoltre un diverso problema: essendo molto vario il rapporto larghezza-altezza dei diversi terminali grafici, se si disegna un cerchio e si hanno window e viewport quadrate può succedere che si ottengano splendidi esempi di ellissi di ogni eccentricità. Per ovviare tale problema occorre predisporre device drivers che estraggono l'area visualizzabile dall'intervallo ([0,1] x [0,1]), risolvendo con un certo grado di arbitrarietà il clipping rettangolare.

Da un lato la non stretta collaborazione tra propositori di standard, costruttori di sistemi grafici e utenti finali, dall'altro la grande diversità e varietà di applicazioni grafiche e di architetture sviluppate, hanno portato a sminuire notevolmente la possibile efficacia di una standardizzazione: forse l'unica possibilità di ottenere per il futuro risultati migliori è impostare una definizione della capacità di un sistema limitandosi a un solo settore applicativo, cercando di prevenire le tecnologie che si instaurerano e le conseguenti prerogative dei sistemi delle nuove generazioni, e trovando di conseguenza un naturale accordo con gli stessi produttori.

6. RIFERIMENTI

[1] "STATUS REPORT OF THE GRAPHIC STANDARD PLANNING COMMITEE", Computer Graphics, A quarterly Report of Sigraph-ACM, vol. 13, n. 3, 1979

[2] R. Eckert et al., "GKS '79: PROPOSAL OF A STANDARD FOR A GRAPHICAL KERNEL SYSTEM", Proceedings of Eurographics 79, Bologna, 1979

[3] A. Granelli, D. Marini, S. Missaglia, "UNA VERSIONE DEL SIGGRAPH CORE IN AMBIENTE UCSD PASCAL", Atti Terzo Congresso AICOGRAPHICS 83, Milano, 1983

7. NOTA BIBLIOGRAFICA

In Italia viene pubblicata in sei numeri annuali la rivista PIXEL.
La rivista presenta contributi scientifici e tecnici originali e di rassegna nei campi della Computer Graphics, della elaborazione di immagini, del CAD nei diversi settori.

Altre pubblicazioni a diffusione internazionale sono:

- COMPUTER GRAPHICS AND APPLICATION - IEEE, trimestrale
- COMPUTER GRAPHICS - ACM SIGGRAPH (Special Interest Group on C.G.), trimestrale
- COMPUTER AND GRAPHICS - Pergamon Press, trimestrale specialmente orientato all'ambiente scientifico e di ricerca.
- ACM TRANSACTION ON COMPUTER GRAPHICS - edito dall'ACM, trimestrale.

Convegni di rilievo su temi della C.G. sono in Italia, il Congresso annuale della AICOGRAPHICS, (associazione italiana di Computer Graphics) che si svolge solitamente in novembre; a livello Euopeo il congresso Eurographics (settembre). Di estrema importanza i due congressi in USA: SIGGRAPH e NCGA, che si svolgono in estate e si contendono il primato di partecipazione.

UN EDITOR PER RETI DI PETRI IN UNA METODOLOGIA DI COSTRUZIONE DI SISTEMI COMPLESSI

E. Ciapessoni, M. Negri
Istituto di Cibernetica - Università degli Studi di Milano

ABSTRACT

This paper describes a Petri net editor based on principles of interactive computer graphics. The editor works with diagrams and texts, which describe, in the usual manner, the formal content of the net, and produce a data structure for the elaboration of properties of the model.
The implementation algorithms are useful in the methodological analysis of complex systems.

RIASSUNTO

Viene descritto un editor per reti di Petri basato sui principi della Interactive Computer Graphics e su alcune particolari forme di interazione per rendere il programma estremamente flessibile. L'editor lavora con diagrammi che rappresentano graficamente reti di Petri e con una struttura dati per l'elaborazione del contenuto formale della rete. Il lavoro risulta, a nostro avviso, adatto come strumento per la progettazione, lo studio e l'analisi di sistemi complessi.

1. INTRODUZIONE

In questo articolo viene presentato un sistema grafico interattivo per la costruzione e l'analisi di reti di Petri, realizzato nell'ambito di un progetto sperimentale denominato "Laboratorio Reti di Petri" dell'Istituto di Cibernetica dell'Università degli Studi di Milano.

Il sistema consente di costruire, modellare e studiare con semplicità anche reti di Petri notevolmente complesse. Tale semplicità è stata ottenuta grazie alla flessibilità delle sue funzioni, alla completezza delle primitive di manipolazione grafica, a una collezione di procedure per lo studio delle reti, ma soprattutto grazie alla sua estrema interattività, basata sulla possibilità di avere a disposizione, in qualunque fase del lavoro, tutte le funzioni esistenti senza interrompere l'operazione in atto.

2. RETI DI PETRI

Le Reti di Petri sono uno strumento logico matematico di tipo grafico che permette di studiare, descrivere e simulare sistemi complessi.

Il pregio delle reti di Petri è di associare al rigore matematico la chiarezza del liguaggio grafico.

Esistono diversi tipi di reti [1]: le reti Condizioni Eventi (CE), le reti Posti Transizioni (PT), le reti Predicati Eventi (PE), le reti di Relazione (R), ecc., che si differenziano per complessità e per capacità di modellare sistemi.

Indipendentemente dal loro tipo, le reti di Petri sono costituite da:

POSTI: rappresentano risorse o condizioni e sono solitamente indicati con dei cerchi;

TRANSIZIONI: rappresentano azioni o eventi e sono solitamente indicati con dei rettangoli;

ARCHI: rappresentano legami logici o fisici tra posti e transizioni; gli archi vengono indicati con delle linee che uniscono fisicamente due oggetti e con frecce che ne indicano il verso.

3. DESCRIZIONE DEL SISTEMA

Si tratta di un programma costruito per la manipolazione grafica e strutturale delle reti di Petri nei suoi tipi principali (CE, PT, PE, R).

Gli obiettivi perseguiti in questo lavoro sono fondamentalmente tre.

Il primo è stato il tentativo di costruire un ambiente che permette all'utente di usare una qualunque metodologia di lavoro sia per la descrizione grafica sia per quella strutturale.

In altre parole, non esiste nessun ordine prefissato di azioni: ad esempio l'utente, dopo aver costruito un oggetto, può scegliere di descriverlo subito o in un secondo momento; inoltre l'oggetto, entro certi limiti, assume caratteristiche grafiche, ma anche strutturali, scelte dell'utente stesso e da lui sempre modificabili.

Più in generale le caratteristiche grafiche e strutturali possono riguardare un sistema arbitrario di oggetti o anche la totalità di essi.

Il secondo obiettivo è stato quello di fornire all'utente un insieme di funzioni ampio, completo e, soprattutto, razionalmente organizzato.

Infine si è curata particolarmente la interattività, che deve rendere il programma efficiente e al contempo facile da usare e da imparare; anche chi ha una conoscenza superficiale delle reti di Petri può usare questo programma al più assistendo a qualche breve "esempio d'uso" senza nessun corso o preparazione specifica.

Per quanto riguarda le caratteristiche tecniche, queste dipendono soprattutto dal non aver imposto alcun limite al numero di oggetti, derivante dall'adozione di strutture dati statiche: il sistema utilizza unicamente strutture dati dinamiche organizzate in un grafo.

Il programma è stato scritto in Pascal, ed è composto dai seguenti moduli: struttura dati, gestione video alfanumerico, gestione video grafico, gestione del disco, manipolazione delle variabili di sistema, manipolazione strutturale degli oggetti, manipolazione grafica degli oggetti, analisi della rete.

Tutte le funzioni strettamente legate alla macchina sono opportunamente isolate ed interfacciate: è già stato possibile trasferire il programma su altre due macchine sostanzialmente differenti in tempi soddisfacenti.

Il programma sarà disponibile in varie versioni che necessitano di 64kbyte (con programmazione segmentata), e al massimo 256kbyte (in programmazione non segmentata); una unità plotter o una stampante grafica. È inoltre consigliabile una unità grafica interattiva (mouse, tablet).

4. INTERFACCIA UTENTE

All'inizio della sessione di editing lo schermo si presenta diviso in quattro aree: (fig. 1).

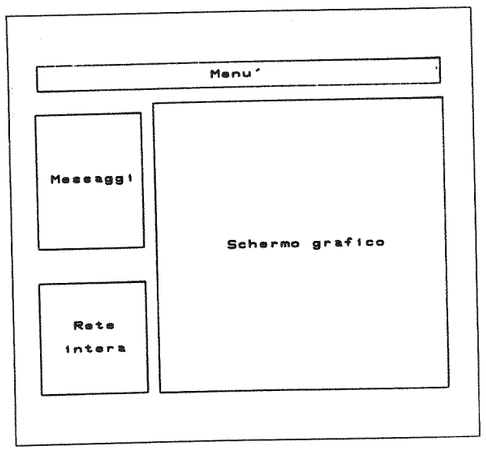


Fig. 1

(1) un menù principale, di stile analogo a quello del sistema UCSD Pascal, in base al quale si possono scegliere le varie attività o comandi;

(2) un'area per il dialogo con l'utente e per le informazioni sullo stato del sistema, contenente il comando corrente, lo stack dei comandi ed altri messaggi;

(3) due finestre grafiche per la costruzione della rete: la prima è l'area di lavoro principale. in cui compare la vista corrente della rete;
la seconda dà una visione della rete nel suo complesso, rappresentandola molto schematicamente: questa individua unicamente i posti e le transizioni della rete e la posizione della vista corrente.

Per non limitare la dimensione massima della rete non si è imposto alcun limite logico alla pagina grafica di lavoro.

È possibile quindi costruire una rete molto estesa ricorrendo a viste parziali della rete nello schermo di lavoro principale.

Sono stati quindi implementati una serie di comandi per ingrandire, ridurre e spostare una rete al fine di consentire la visualizzazione più opportuna in ogni momento.

La costruzione di una rete per raffinamenti successivi o per composizione di blocchi di rete (top-down o bottom-up) è uno strumento importante e utile per la descrizione di sistemi di dimensioni considerevoli. Per questo motivo il Net Editor dà la possibilità di utilizzare i morfismi e gli antimorfismi propri delle reti di Petri.

Le attività di editing della rete possono essere divise in:

1) attività di creazione e/o modifica grafica e strutturale della rete;

2) attività di modifica grafica rappresentativa;

3) attività di modifica dell'oggetto corrente;

4) attività di gestione dell'I/O.

Le attività di creazione/modifica riguardano la costruzione della rete, mentre le attività di modifica rappresentativa permettono di ottenere viste differenti della rete e di muoversi all'interno della struttura ad albero dei raffinamenti.

Per semplificare al massimo l'interazione con l'utente si è progettato il sistema in modo da permettere una gestione ricorsiva delle varie attività: con questa tecnica si ha la possibilità di interrompere una attività in corso per riprenderla in un momento successivo,

con un ambiente di costruzione omogeneo e con i medesimi comandi in ogni momento.

Per avere una interazione con l'utente immediata nella fase di creazione di un oggetto o di modifica delle sue caratteristiche grafiche si è adottata la tecnica nota col nome di *dragging*. Questa consiste nell'adattare la forma del cursore grafico all'operazione in corso; ad esempio, nella modalità di creazione-transizioni il cursore è rappresentato da un rettangolo, mentre in quella di spostamento-oggetti il cursore assume la forma dell'oggetto stesso.

Gli oggetti possono essere identificati singolarmente o collettivamente rispetto ad una finestra, ciò per permettere sia operazioni dettagliate sul singolo oggetto che veloci operazioni su gruppi di oggetti.

5. COMANDI DI SISTEMA

Il menù principale presenta le seguenti attività:

1. attività di creazione-modifica della rete:

creazione cancellazione
riposizionamento copia
editing

2. attività di modifica rappresentativa:

zoom in out
raffinamento contrazione
spostamento

3. attività di modifica degli oggetti di riferimento e di alcune variabili di sistema:

scelta ritorno

4. gestion dell'I/O:

buffer-comandi richiamo-reti
stampa usciata

5. analisi della rete:

manuale random
storia

Per mantenere l'omogeneità del sistema nel suo complesso i vari livelli dei menù sono strutturati secondo la filosofia del sistema operaivo UCSD Pascal cioè con dei menù ad albero che compaiono nella parte superiore dello schermo e dei quali è possibile richiamare una particolare funzionalità tramite la pressione di un singolo tasto.

Nel seguito viene presentata una descrizione più appropriata dei comandi implementati per dare un visione globale delle possibilità offerte dal sistema.

In fig. 2 e fig. 3 esempi di output grafico ottenuto con il sistema presentato.

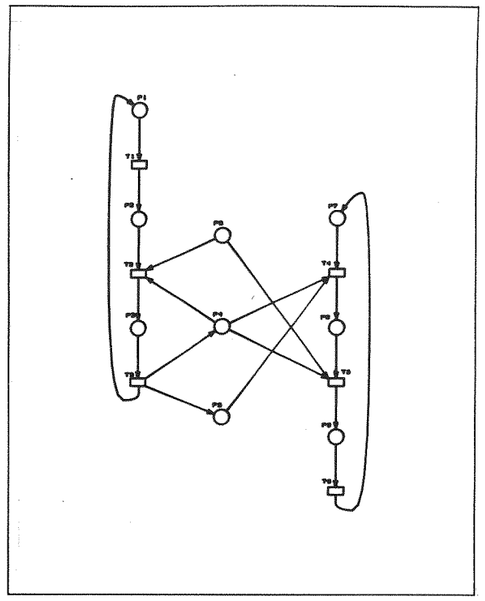


Fig. 2

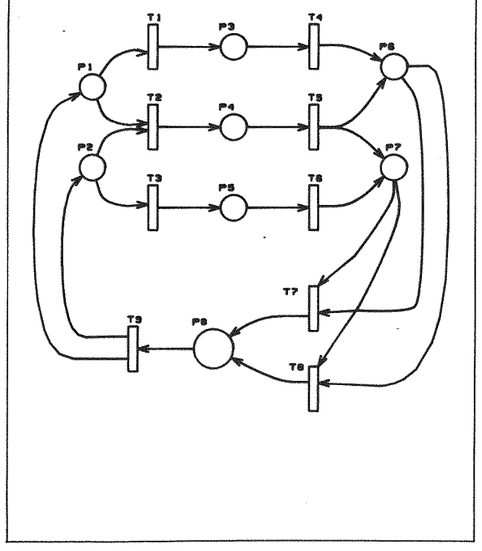


Fig. 3

- **Creazione**

L'attività di creazione consente di inserire nuovi oggetti elementari. Gli attributi grafici degli oggetti creati dipendono da un modello di riferimento corrente, eventualmente modificabile mediante un comando, che permette di fissare le dimensioni, il colore, il tipo di linea dell'oggetto di riferimento.

Per aumentare l'interattività si è fatto sì che il cursore assuma la forma dell'oggetto di riferimento, in modo da mostrarne direttamente l'occupazione grafica nell'ambito della rete complessiva e di avere degli "ambienti" di creazione diversificati immediatamente riconoscibili.

Nella creazione di un posto o di una transizione, il cursore assume perciò la forma e le dimensioni dell'oggetto tipo; invece, nel caso dei connettori assume la forma di un "elastico" allungabile che segue gli spostamenti della penna della tavoletta grafica.

In questa attività è inoltre possibile creare dei raffinamenti, cioè prendere una parte della rete, specificata mediante una opportuna finestra, e metterla in un sottolivello legato ad un oggetto; ciò è utile per strutturare una rete costruita con un approccio più tradizionale, per selezionare una particolare sottorete o per costruire una banca dati di sottoreti.

- **Cancellazione**

L'attività complementare a quella di creazione è quella di cancellazione degli oggetti, delle sottoreti o dei morfismi.

Oltre alla cancellazione di un singolo oggetto è disponibile la cancellazione mediante una finestra, che coinvolge tutti gli oggetti contenuti all'interno di un'area dello schermo grafico, fornendo un metodo flessibile di interazione.

Risulta poi utile disporre di un comando di cancellazione dei raffinamenti in modo da poter portare una sottorete al livello superiore.

- **Riposizionamento**

Un'altra attività di editing inserita nel Net Editor è quella di modifica delle caratteristiche grafiche degli oggetti della rete come il colore , la dimensione la posizione ecc.

Ciò è ottenibile tramite il comando di riposizionamento, che permette di effettuare modifiche posizionali e/o dimensionali degli oggetti della rete, singolarmente o in gruppi (identificati tramite una finestra) e di modificare gli attributi di un oggetto esistente prendendo a modello l'oggetto di riferimento.

- **Copia**

Per completare le attività di creazione e modifica della rete è stato inoltre implementato un comando di copia: di un singolo oggetto, di una sottorete o degli attributi grafici di un oggetto; sostanzialmente, si ha una quasi completa corrispondenza con il comando di riposizionamento, salvo per il fatto che gli oggetti sono fisicamente duplicati.

Mediante questo comando è possibile inoltre gestire un menù di forme tipiche per eliminare la necessità di effettuare continue modifiche agli oggetti di riferimento sia per la creazione di oggetti, sia per la modifica degli attributi di questi.

Anche in questa attività si è utilizzato, per quanto possibile la tecnica del dragging degli oggetti, in modo da dare una visione il più possibile precisa dell'attività in corso.

- **L'editing**

L'editing delle caratteristiche strutturali della rete viene effettuato tramite il comando di editing che consente di inserire o modificare la capacità dei posti; il peso degli archi, la marcatura della rete, tutti concetti che dipendono strettamente dal tipo di rete in costruzione. Nel caso di reti CE/PT si ha che la capacità dei posti e il peso degli archi è un numero intero, mentre nelle reti PE/R la capacità è un vettore.

Per permettere questo editing strutturale è stato implementato un "editor nell'editor" che si presenta in modo trasparente all'utente e che gestisce le strutture dati dinamiche associate alle strutture matematiche degli oggetti: l'utente, dopo aver scelto il tipo di rete da realizzare viene automaticamente indirizzato nella costruzione della struttura matematica degli oggetti della rete.

La fase di editing strutturale della rete non deve essere effettuata in un momento particolare: l'utente ha la più completa libertà di dare le caratteristiche strutturali di un oggetto nel momento ritenuto più utile.

Una particolare nota riguarda la presenza di un terzo etitor nell'editor, cioè quello che permette di specifirare le matrici associate agli archi nelle reti PE/R.

- **Zoom in out**

I comandi di modifica rappresentativa consentono di visualizzare la rete nella sua globalità o in un suo particolare, per dare la possibilità di lavorare a diversi livelli di dettaglio nella costruzione della rete di Petri o di costruire la rete mediante raffinamenti successivi top-down, o con contrazioni utilizzando sottoreti prese dalla banca delle reti, bottom-up.

In altre parole sono a disposizione 5 tipi di zoom due continui, due discreti e uno che calcola il coefficiente di ingrandimento e lo spostamento necessari alla visualizzazione completa della rete nella finestra grafica.

- **Spostamento**

Sono stati implementati comandi di spostamento nel piano attuale della rete, per i raffinamenti e le contrazioni.

La costruzione di un connettore tra due oggetti può passare, mediante la catasta ricorsiva dei comandi, attraverso una quaunque modifica rappresentativa.
Gli spostamenti avvengono nelle quattro direzioni fondamentali per pagine, per passi discreti modificabili dall'utente o in base ad un vettore grafico che descrive direzione verso e quantità dello spostamento.

- **Raffinamento contrazione**

Per quel che riguarda i raffinamenti e le contrazioni (morfismi e antimorfismi di Reti di Petri) questi sono stati considerati come la possibilità di un oggetto di contenere a sua volta una rete che può essere legata con l'esterno o direttamente tramite dei connettori o indirettamente tramite porte; proprio al fine di meglio rendere graficamente questo aspetto a più piani delle reti di Petri è in preparazione una versione tridimensionale del Net-Editor.

Da un punto di vista grafico i morfismi consistono nell'"apertura" di un nuovo piano grafico legato ad un oggetto e gli antimorfismi nella risalita al piano superiore.

- **Scelta**

La scelta della forma e dimensioni degli oggetti di riferimento e di una serie di condizioni di funzionamento del sistema nel suo complesso può essere effettuata mediante il comando di modifica dell'oggetto corrente (setting).

Se necessario è poi possibile introdurre dei controlli "sintattici" sulla costruzione effettuata, per verificarne la consistenza e per impedire collegamenti privi di senso tra gli oggetti della rete.

Vi sono delle scelte di default del sistema riguardo a questi comportamenti che possono essere cambiate, oltre che con il comando di scelta, anche con l'utilizzo di un file di configurazione che contiene un buffer di comandi. In questo modo l'utente ha la possibilità di ottenere un ambiente personalizzato alle proprie esigenze.

Per quanto riguarda la gestione ricorsiva dei comandi, è stata introdotta una distinzione tra i comandi di creazione-modifica della rete e i comandi di modifica rappresentativa o di scelta: i primi, per terminare, necessitano di un "escape" esplicito, invece i secondi hanno una uscita immediata al termine dell'azione ad essi associata.

Questa leggera "antisimmetria" tra i comandi è stata ritenuta utile in quanto i comandi del secondo tipo non necessitano generalmente di essere ripetuti, ma vengono usati per una azione immediata.

- **Analisi**

L'analisi della evoluzione dinamica della rete può essere effettuata o manualmente o automaticamente con diversi algoritmi di simulazione che, essendo completamente integrati con il resto del Net Editor, possono essere invocati in ogni fase della costruzione.

La dinamica della rete può essere rappresentata a diversi livelli di dettaglio dati dalla struttura dei raffinamenti: un semplice cambiamento del punto di vista permette di focalizzare l'attenzione su una particolare sottorete della struttura dei raffinamenti stessi o di ingrandire un particolare settore della rete in studio.

Il richiamo della "storia" della simulazione, mediante il meccanismo di memorizzazione delle transizioni scattate, consente di rivedere lo svolgimento dell'ultima analisi effettuata, eventualmente modificando prima il punto di vista in modo da poter osservare una sequenza di eventi su parti separate della rete.

Da ultimo l'uscita dal sistema è strutturata secondo la filosofia del sistema operativo UCSD Pascal, con la possibilità di salvataggio della rete su un file per il suo utilizzo per una successiva fase di editing o di analisi.

6. SVILUPPI

Sono in corso ulteriori sviluppi del NET EDITOR consistenti in una serie di funzioni per la visualizzazione e l'eleborazione tridimensionale delle reti; è inoltre in fase di progetto la possibilità di associare un linguaggio interattivo agli eventi delle reti per potenziare le regole di scatto alle transizioni utilizzando anche espressioni logico-matematiche, che consentono un maggior potere espressivo della modellazione di sistemi complessi.

7. CONCLUSIONI

È stato progettato e sviluppato un editor grafico interattivo per la costruzione di reti di Petri strutturato in modo da poter essere utilizzato come strumento per l'analisi di sistemi concorrenti.

Esiste infatti l'esigenza di uno strumento per la progettazione e per la verifica della correttezza di sistemi complessi: sistemi operativi, centrali telefoniche, sistemi real time ecc., che sempre più necessitano di una metodologia di progetto che consenta la validazione e l'analisi delle scelte progettuali effettuate.

Nella sua realizzazione sono state utilizzate le tecniche della Interactive Computer Graphics e si è ottenuto un sistema di notevole flessibilità e facilità d'uso, grazie soprattutto alla gestione ricorsiva dei comandi e alle modalità di visualizzazione adottate. Si è, in altre parole, indirizzato lo sforzo alla creazione di un programma a misura d'uomo e non di un uomo a misura di macchina.

8. RINGRAZIAMENTI

Numerosi e costruttivi sono stati i consigli e le critiche giunti dall'Istituto di Cibernetica dell'Università di Milano che molto hanno contribuito alla realizzazione del sistema: ringraziamo il Prof. G. Degli Antoni e il Prof. D. Marini per i consigli dati durante le fasi di analisi del lavoro; ringraziamo inoltre, per la loro pazienza, gli utenti del Net Editor che hanno condotto la sperimentazione sul campo e Daniele Pieragostini, autore degli algoritmi di analisi implementati nell'ambito del Laboratorio Reti di Petri.

Il progetto si è avvalso di un elaboratore HP9836 con una serie di periferiche, gentilmente fornito dalla HP Italiana, alla quale va il nostro ringraziamento.

Questo lavoro si svolge nell'ambito del progetto CP di collaborazione tra la HIS Italia e l'Istituto di Cibernetica dell'Università di Milano.

9. BIBLIOGRAFIA

[1] Reisig, PETRINETZE Springer-Verlag, 1982

[2] H. Genrich, R. Shapiro, A DIAGRAM EDITOR FOR LINE DRAWINGS WITH INSCRIPTIONS, Varenna, 1982

[3] Beaudouin, Lafon, PeTriPote, A GRAPHIC SYSTEM FOR PETRI-NETS DISIGN AND SIMULATION, Fourth European Workshop on Applications and Theory of Petri Nets

SOFTWARE SCIENCE: UNA METRICA PER IL SOFTWARE

Emilio Bertolotti
HISI - Pregnana Milanese

RIASSUNTO

Nella nota vengono presentati i fondamenti teorici e alcuni tra i più significativi risultati sperimentali della "software science".
La teoria, nata da un'idea di M. Halstead, si prefigge di stabilire le relazioni in grado di quantificare diverse ed importanti proprietà degli algoritmi come la complessità, intesa come difficoltà di comprensione, il tasso di errori presenti in un'implementazione o il tempo di sviluppo. Lungi dall'avere subito, anche solo temporaneamente, un assestamento, la "software science" è tutt'ora continuamente in evoluzione e comunque sempre al centro di un dibattito che vede interessati sia oppositori che sostenitori, le opinioni dei quali sono sinteticamente esaminate nel corso di questa breve rassegna.

ABSTRACT

The theoretical foundaments and some of the most significant experimental results of the "Software Science" are presented.
The major goal of the theory developed by M. Halstead is the quantification of important algorithm "measures" such as "complexity", intended as difficulty of comprehension "error rate", "development time".
The most important aspects and results of the theory are investigated.

1. INTRODUZIONE

Non più di dieci anni or sono, M. Halstead, un ricercatore della Purdue University, di recente scomparso, ebbe la singolare idea di investigare, con un approccio tipico delle scienze naturali, su ciò che costituisce una delle fasi più importanti della vita di un programma: la "nascita". Egli volle cioè descrivere l'attività di implementazione di un programma analizzandola con il metodo tipico della scienze naturali basato sulla conferma sperimentale "a posteriori" delle ipotesi formulate.

Questa originale idea stimolò lo sviluppo di una serie di lavori a cura di Halstead stesso ed altri, che culminarono nel 1977 con la pubblicazione del testo "Élements of Software Science", Elsevier 1977, M. Halstead.

A tale testo si rimanda chi volesse una dettagliata ed esauriente trattazione dell'argomento che viene solo sinteticamente presentato tralasciando, oltre alle dimostrazioni matematiche, quelle parti della originale formulazione di Halstead ritenute di minor importanza.

L'obiettivo principale di questa relativamente giovane teoria, che va sotto il nome di "Software Science", consiste nella formulazione di un modello in grado di quantificare e correlare tra di loro proprietà del tutto generali dei programmi quali la lunghezza, la complessità, il tempo necessario per l'implementazione, ecc...

Halstead, con un approccio ipotetico-deduttivo, affronta l'argomento individuando tre passi fondamentali da sviluppare:

1) denifizione delle grandezze osservabili in software-science;

2) formulazione delle ipotesi di relazioni esistenti tra le osservabili definitite;

3) verifica sperimentale delle relazioni ipotizzate.

Rispettando tale successione illustriamo punto per punto le tappe fondamentali della teoria.

2. DEFINIZIONI DELLE OSSERVABILI

In ogni implementazione di un algoritmo in un qualsiasi linguaggio è possibile identificare tutti gli *operandi,* definiti come variabili o costanti che vengono usati. È anche possibile identificare tutti gli *operatori* definiti come simboli o combinazioni di simboli che modificano il valore (es.: "+") di un operando o prescrivono la sequenza di azioni o di valutazioni da eseguire (parentesi, istruzioni di controllo,...).

Oltre agli articoli referenziati nei richiami bibliografici e ai lavori di M. Halstead, che costituiscono la maggior base di indagine a cui attinge questa rassegna, alcune considerazioni sviluppate in questa nota sono tratte anche dalle esperienze accumulate nella messa a punto di una procedura di conteggio operatori/operandi per programmi Pascal. Tale lavoro è stato svolto con la collaborazione degli studenti di TAMC del corso di laurea in scienze dell'informazioe, di M. Ferrari e M. Ornaghi dell'Istituto di Cibernetica della Università degli Studi di Milano. Si ringraziano inoltre G. Degli Antoni e M. Ferrari per il loro apporto determinante alla discussione sul tema trattato.

L'identificazione di operatori e operandi permette di definire un insieme di entità misurabili, presenti in ogni versione di un dato algoritmo:

n_1 = numero di *operatori* distinti che compaiono in quella implementazione.

n_2 = numero di *operandi* distinti

N_1 = numero totale di *operatori.*

N_2 = numero totale di *operandi.*

n_2^* = numero di *operandi* di input/output che compaiono nella specifica.

Si veda l'esempio di programma Pascal in figura 1.

Definiamo queste cinque quantità direttamente misurabili per conteggio in una qualsiasi implementazione, **osservabili fondamentali** della software science.

A partire dalle osservabili fondamentali, è possibile definire altre osservabili, che caratterizzano una data implementazione, quali ad esempio la lunghezza del programma o il vocabolario usato. Chiameremo tali quantità **osservabili non fondamentali** o semplicemente **proprietà** di un dato programma, essendo grandezze che delineano alcune caratteristiche, particolari di quel programma.

Numerose e varie sono le proprietà di un programma definibili a partire dalle osservabili fondamentali, ma solo alcune di queste sono di peculiare interesse per la software science.
Vediamo quindi quali sono le proprietà che assieme alle osservabili fondamentali formano l'insieme delle quantità che ci interessa misurare in un qualsiasi programma.

- **Il vocabolario n**

Con vocabolario n di un programma si intende la somma tra il numero di operatori distinti ed il numero di operandi distinti di cui si avvale il programmatore per l'implementazione:

$$n = n_1 + n_2$$

Si noti che n_1, ossia il numero di operatori distinti, è in generale un sottoinsieme del numero di operatori che il linguaggio di programmazione usato mette a disposizione.

- **La lunghezza N**

Definiamo lunghezza N di un programma la quantità:

$$N = N_1 + N_2$$

dove N_1 e N_2 sono, come già detto, il numero di occorrenze di operatori ed operandi.

SPECIFICHE: sommare una quantità n di numeri positivi e stampare il risultato della somma e degli eventuali numeri negativi.

IMPLEMENTAZIONE:

```
program sumpositivi (input, output);
    var i,n,s, : integer,
        x      : array [1..100] of integer;
    procedure sum;
        var c: integer;
            begin
                s:= 0;
                for i:=1 to n do
                    if x [i] › 0 then s:= (s+x[i])
                    else c:= c + 1;
                write ('val. negativi:'.c)
            end;
begin
    read (n),
    for i:= 1 to n do read (x[i]);
    sum;
    writeln ('somma:',s)
end.
```

TABELLA OSSERVABILI FONDAMENTALI

(Gli apici sono usati come metasimboli delimitatori)

OPERATORI:	**'begin-end'** 'write' '()' **'for'** ';' '+' **'if'** ';' '[]' ':=' 'read 'writeln' 'sum'
OPERANDI:	'c' 'i' 'n' 's' 'x' '1' 'Ø'
OSSERVABILI:	n_1=13, n_2=7, N_1=30, N_2=23, n^*_2=(n,s,x,c,)=4 (i conteggi sono stati effettuati escludendo le dichiarative di dati, di programma e di procedura)

Fig. 1

- **Il volume**

Dato un vocabolario $n = n_1 + n_2$ è possibile rappresentare ciascun operando o operatore in modo univoco mediante una stringa di cifre binarie. Il numero necessario e sufficiente di variabili binarie (bit) per poter rappresentare senza ambiguità ciascun elemento del vocabolario n è dato da:

$$numbit = \log_2 n$$

Quindi indipendentemente dal linguaggio di programmazione usato e dal formalismo di rappresentazione degli operandi, numbit da il minimo numero di simboli necessari per rappresentare completamente e senza ambiguità ciascun elemento del vocabolario n.

Definiamo allora **volume di un progaramma** la quantità:

$$V = N \log_2 (n_1 + n_2)$$

Esso ci da una misura in bit delle "dimensioni" di un programma.

È universalmente accettata la tesi secondo cui ogni programma, comunque complesso, può essere espresso nella forma di una chiamata ad una funzione più una lista di operandi. Si pensi al caso del richiamo di una "procedura" o al richiamo di una subroutine, via istruzioni di "call", nei linguaggi di programmazione più diffusi.
Più precisamente, definiamo "Forma minimale" di un dato progaramma la scrittura dello stesso che impiega: un operatore di chiamata alla funzione, un operatore di assegnamento o raggruppamento di operandi, più gli operandi che vengono passati alla funzione richiamata.
Consideriamo il caso del calcolo del numero di Fibonacci fatto da una procedura che riceve il valore di input in INNUM e calcola l'output in OUTFIB.
In questo caso si piò scrivere:

FIBONACCI (INNUM, OUTFIB)

ed analizzando tale scrittura si possono individuare appunto:

operatori: "FIBONACCI", "(,)"
operandi: "INNUM" , "OUTFIB"

quindi:

$$n_1 = N_1 = 2$$
$$n_2 = N_2 = 2$$

inoltre il numero di operandi di input/output n_2^* è proprio uguale al numero di operandi singoli n_2. Se ne deduce quindi che un programma espresso nella forma "minimale" è scritto in modo che la sua lunghezza N coincida con il vocabolario, ossia per ogni simbolo del vocabolario esiste una ed una sola occorrenza. In breve, il programma è scritto con il minor numero possibile di operatori e operandi.

È da notare che per qualsiasi forma minimale il numero di operatori è sempre uguale a due.

Definiamo **volume potenziale** V^* il volume di un programma scritto nella sua forma minimale:

$$V^* = (2+n^*_2) \log_2 (2+n^*_2)$$

Vale la relazione:

$$V \geq V^*$$

e varrà in particolare il segno di uguale quando esiste l'operatore ad hoc nel linguaggio di programmazione prescelto, in grado di soddisfare da solo le richieste contenute nell'algoritmo da implementare. Si pensi ad esempio al seguente triviale programma Pascal:

"si sommino in C i due reali A,B"
C:= A + B

applicando le definizioni generali si ha:

$$V^* = V = 5$$

infatti $n_1 = N_1 = 2$ ed anche $n_2 = = N_2 = 3$.

- **Il livello di programma L; il livello di linguaggio** λ

Mentre i concetti intuitivi di livello di linguaggio e livello di un programma sono tanto diffusi da entrare a far parte del gergo comune per gli addetti ai lavori, non altrettanto diffuso è il loro utilizzo ingegneristico formale, in quanto restano quantità definite in modo aleatorio e tali da rendere altamente soggettiva ogni valutazione sul loro conto.

Halstead cerca di superare questa arbitrarietà di definizione proponendo le quantità misurabili **livello di programma L** e **livello di linguaggio** λ come:

$$L = V^*/V$$
$$\lambda = LV^*$$

La prima relazione quantifica il livello di difficoltà (di comprensione) di un programma; 1/L cresce quanto più il programma aumenta di volume rispetto alla sua forma minimale scritta in linguaggio potenziale ed è generalmente accertato quanto siano in generale più difficili da leggere ed interpretare programmi più voluminosi *relativamente alle specifiche.*

Si pensi, ad esempio, ad un programma Pascal che esegue il prodotto di due matrici e che abbia un volume V_1 ed un livello L_1. Riscrivendo lo stesso programma in assembler, a causa della povertà dei tipi di dati che tale linguaggio mette a disposizione, misureremo un volume $V_2 \rangle V_1$ e quindi $L_1 \rangle L_2$ essendo V^* sempre lo stesso in entrambi i casi (stesse specifiche).

L rappresenta quindi un relazione che ci dà un'indicazione precisa del fatto che il particolare algorit-

mo scelto, codificato in Pascal, è più comprensibile, ossia ha un livello di programma maggiore, che lo stesso algoritmo implementato in assembler.

Consideriamo invece programmi con V* diversi scritti nello stesso linguaggio; è allora ragionevole pensare che al crescere di V*, il volume V cresca più rapidamente e che quindi L, dato dal rapporto tra i due, diminuisca. Halstead ipotizza che la crescita di V* sia direttamente proporzionale al calo di L considerando quindi $\lambda = LV^*$ una costante di linguaggio.

Quindi il valore numerico λ quantifica il livello del linguaggio nel senso che linguaggi "più potenti", ossia potenzialmente più espressivi, produrranno, a parità di V*, programmi con volumi V minori e quindi con livelli L maggiori.

In sostanza, dato un algoritmo con volume potenziale V* e le relative implementazioni nei linguaggi λ_1 e λ_2 con $\lambda_1 < \lambda_2$, allora troveremo che l'implementazione che fa uso del linguaggio λ_1 è caratterizzata da un volume $V_1 > V_2$ e conseguentemente $L_1 < L_2$.

Considerazioni ulteriori derivate da misure con programmi scritti in diversi linguaggi mostrano che λ ha una notevole varianza e che ciò che può essre assunto come il livello di un linguaggio è in realtà il valor medio di λ piuttosto che un singolo valore del prodotto LV*.

Quindi:

$$\lambda = \sum_{1}^{m} L_i.V^*_i/m$$

- **Lo sforzo**

Con la definizione dell'osservabile E (Effort) ossia lo **sforzo di programmazione,** Halstead vuole dare una quantificazione dell'attività mentale richiesta al programmatore per l'implementazione di un algoritmo.

Egli tenta di dare un interpretazione dello sforzo compiuto dal programmatore in termini di numero di discriminazioni mentali necessarie per l'implementazione di un programma di lunghezza N, avendo a disposizione un vocabolario di n elementi.

Supponendo quindi che la costruzione di un tale programma implichi il dover compiere N volte una ricerca di tipo dicotomico in una tabella ordinata di n elementi, allora ricava la relazione che definisce lo sforzo E come:

$$E = V/L$$

(Per la dimostrazione si veda [1]).

Lo sforzo di programmazione E risulta quindi crescere in modo direttamente proporzionale alle dimensioni del programma V e al suo grado di complessità 1/L.

Halstead ipotizza quindi che la grandezza misurabile E quantifichi lo sforzo di programmazione e sia correlata, come vedremo in seguito nella trattazione sulle relazioni tra le osservabili, al tempo di sviluppo di un programma. (Qualcuno preferisce adottare E piuttosto che 1/L come indice di complessità di un programma. Si noti che comunque le due grandezze devono avere lo stesso andamento; comunque variino V e V*, allora L ed E o crescono o diminuiscono entrambe).

3. RELAZIONI TRA LE OSSERVABILI

Accanto alle precedenti definizioni delle osservabili (n_1, n_2, N_1, N_2, n^*_2, N, V, V*, L, λ, E) Halstead ipotizza l'esistenza di alcune relazioni che legano tra di loro tali quantità.

I presupposti teorici che stanno alla base della formulazione di tali relazioni non sono sempre rigorosamente dimostrabili e Halstead stesso propone la verifica sperimentale come migliore strumento per accertare la correttezza delle sue ipotesi intuitive.

- **L'equazione della lunghezza**

Fra le più importanti relazioni, quella che lega la lunghezza N al numero di operatori distinti n_1 e al numero di operandi distinti n_2 è senza dubbio una tra le più sperimentate; essa afferma che la lunghezza N di un qualsiasi programma è ricavabile dalla seguente equazione:

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

Le ipotesi che hanno condotto Halstead alla formulazione di questa ed altre relazioni sono contenute nel testo [1] e verranno omesse in quanto riteniamo più significativo cercare di presentare e commentare gli esperimenti di verifica fin qui condotti.
In [1] Halstead riporta un esperimento di verifica eseguito sui primi 14 algoritmi pubblicati dall'ACM trovando che $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$ è una buona stima della lunghezza misurata $N = N_1 + N_2$. Riproduciamo in figura 2 la tabella con i valori di $\hat{N}$ calcolati e N misurati.
Anche J.H. Elshoff ottiene risultati soddisfacenti [5] compiendo misure su programmi di tipo commerciale. Altre misure di programmi FORTRAN a sostegno dell'equazione della lunghezza sono esibiti in [13].

Algoritmo	N	$\hat{N}$
1	104	104
2	82	77
3	453	300
4	132	139
5	123	123
6	98	101
7	59	62
8	131	117
9	314	288
10	46	52
11	53	52
12	59	62
13	59	57
14	186	163

Valor medio di $N-\hat{N}$: 14,3
Valor medio di N : 135.6

In tutti questi lavori appare chiaro che questa relazione, così come le altre in software science, è valida statisticamente e non singolarmente per ogni programa. Questo è imputabile alla presenza di fattori non deterministici che invevitabilmente appaiono quando si ha a che fare con l'analisi di fenomeni che sono il frutto di un'attività umana quale è appunto il processo di implementazione di un programma.
In breve, ciò suggerisce di considerare $\hat{N}$ una buona stima di N quando si considerino in senso statistico un discreto numero di programmi mentre per singoli casi possono verificarsi notevoli differenze [10].

Più recenti risultati sperimentali [7], [8], [10] evidenziano, per il PL/S [10], come al crescere delle dimensioni effettive di un programma il numero di operatori distinti n_1 non cresca anch'esso, ma tenda ad un valore costante che è il massimo numero di operatori distinti che il linguaggio di programmazione mette a disposizione.

Ciò naturalmente provoca una sottostima di N che sarà tanto più accentuata quanto maggiore sarà la dimensione del programma in questione.

Il superamento del problema suggerito in [10] sembra derivare dal conteggiare come operatori e operandi oltre che quelli presenti nella sezione codice anche quelli presenti negli statment dichiarativi (dati, tipi, ecc.).

In complesso, nonostante la derivazione teorica dell'equazione della lunghezza presenti qualche punto debole [4] [7], il riscontro sperimentale sembra confermare una soddisfacente capacità previsionale almeno per un certo range di dimensioni.

- **$\hat{L}$ come stima del livello misurato L**

Abbiamo già detto di come 1/L possa divenire una misura di complessità di un programma nel senso della "difficoltà" di implementazione e comprensione. È noto come tali caratteristiche incidano rispettivamente sulla quantità di errori introdotti in fase di codifica e facilità di scoperta e rimozione degli stessi durante la manutenzione.

Nel tentativo di ridurre il numero di variabili indipendenti da cui L dipende, Halstead ipotizza che la quantità $\hat{L}$ definita dall'equazione:

$$\frac{1}{\hat{L}} = \frac{n_1}{2} \frac{N2}{n2}$$

sia una buona stima del livello misurato L. Egli riesce quindi sotto tali ipotesi ad eliminare le variabili indipendenti N_1, n^*_2.

Si noti come $1/\hat{L}$, ossia la difficoltà, aumenti con l'aumentare del numero di operatori singoli (n_1) e del rapporto N_2/n_2.

Questo rapporto ci dà un'indicazione di quante volte mediamente venga riutilizzato uno stesso operando all'interno del programma. Più questo numero cresce e più è difficile ricordare o capire il significato e la funzione svolta da un dato operando in un dato punto del programma; si pensi ad esempio ai possibili contenuti di una variabile referenziata più volte.

I risultati sperimentali [8] condotti su eterogenei banchi di test, che mettono a confronto $\hat{L}$ calcolato ed L misurato, non sono certo incoraggianti come facevano invece sperare quelli prodotti da Halstead in [1].

Restano tuttavia soddisfacenti le misure tendenti a correlare la quantità $1/\hat{L}$ con il tasso di errori presenti in un programma. Alcuni rapporti IBM [10], [11] giungono persino a stabilire valori di soglia di $1/\hat{L}$ al di sopra dei quali la manutenzione di un programma diventa critica o addirittura è consigliabile un rifacimento da zero.

Un'altra grandezza, definita da Halstead come il

contenuto di intelligenza di un algoritmo I, è direttamente legata al livello calcolato $\hat{L}$ della seguente equazione:

$$I = \hat{L}V$$

Secondo Halstead, questa quantità dovrebbe essere una invariante di algoritmo, ossia non dovrebbe cambiare per le diverse implementazioni in diversi linguaggi dello stesso algoritmo. Anche qui tuttavia alcuni tra i più significativi test [3] sembrano essere in disaccordo con la presunta costanza predetta dalla teoria; le discordanze già riscontrate su $\hat{L}$ si riperquotono evidentemente anche su I.

- **Lo sforzo E e il tempo di implementazione T**

Nel derivare la definizione di sforzo di programmazione E = V/L, Halstead assume come ipotesi che E rappresenti il numero totale di discriminazioni mentali elementari compiute dal programmatore durante l'implementazione.
Indicando quindi con S la costante di stroud, che dà il numero di discriminazioni mentali elementari per secondo ($5 < S < 18$), è possibile, secondo Halstead, ricavare il tempo necessario per l'implementazione come:

$$T = E/S \text{ (sec.)}$$

Si noti che E e quindi T sono grandezze prevedibili a priori (prima dell'implementazione) in quanto:

$$E = V/L = V^2/V^* = V^{*3}/\lambda^2$$

e sia λ che V^* sono quantità già note già prima che il programma venga scritto.

Una serie di esperimenti [8] confermano, per programmi di piccole dimensioni, una buona correlazione tra il tempo di implementazione previsto T e quello effettivamente misurato. Certo i risultati sono aderenti alle previsioni teoriche, anche in questo caso, solo statisticamente, presentando invece in singoli casi differenze notevoli.

Ciò è imputabile da un lato al largo range di valori ammesso dalla costante di Stroud ($5 < S < 18$), dall'altro al fatto che la relazione T = E/S non tiene conto delle differenze che esistono tra un programmatore esperto ed uno privo di esperienza.

Alcune caratteristiche tipiche del singolo programmatore, che incidono notevolmente sul risultato finale, non entrano quindi come variabili nel computo del tempo di sviluppo T non essendo, almeno per ora, in qualche modo formalizzate.

Halstead in [1] definendo una soglia psicologica di criticità per E ricava una grandezza B ("Bugs") in grado di dare una stima dall'error rate di un programma. Esistono in proposito pochi esperimenti in letteratura.

4. CAPACITÀ PREVISIONALI DELLA TEORIA

Abbiamo fino ad ora esaminato le relazioni ipotizzate da Halstead che già permettono di prevedere alcune grandezze caratteristiche del programma, prima che questo venga scritto. In realtà, come vedremo ora, con le ipotesi fatte è possibile esprimere tutte le osservabili della software science in funzione della due uniche variabili indipendenti λ e V^*: quantità, queste, note già prima che il programma venga scritto.

Senza ulteriori ipotesi, ma semplicemente risolvendo il sistema di equazioni costituito dalla relazioni fino ad ora scritte, si riesce così a corredare la teoria di una completa capacità previsionale che senza dubbio ne costituisce il tratto carattaristico più interessante (anche se resta il meno provato). Si pensi al vantaggio di avere a disposizione uno strumento in grado di predire tempi di sviluppo e occupazioni di memoria di un qualsiasi programma di cui siano note le specifiche.

Purtroppo, le uniche conferme sperimentali di questa presunta capacità della software science vengono da Halstead [1] mentre non esistono, almeno nella letteratura in nostro possesso, altre pubblicazioni che riportino risultati su questo tema.

Veniamo quindi alla formulazione delle equazioni in λ e V^*.
Dal seguente sistema è possibile ricavare per semplice sostituzione le espressioni che danno le osservabili n_1, n_2 in funzione di λ ed n^*_2.

1) $\lambda = LV^*$
2) $L = V^*/V$
3) $V = N \log_2 (n_1+n_2)$
4) $V^* = (2+n_2^*) \log_2 (2+n_2^*)$
5) $V^{**} = (2+n_2^* \log_2 n_2^*) \log_2 (2+n_2^*)$
6) $n_2 = (V^{**}/V^*-1)(n_1-2)+n_2^*$
7) $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$

Le equazione 5 definisce il volume limite V^{**}, quantità ricavata da Halstead sostituendo nell'equazione del volume V, alla espressione della lunghezza misurata N* quella calcolata: $n_1^* \log_2 n_1^* + n_2^* \log_2 n_2^*$ dove si tenga conto del fatto che $n_1^* = 2$ e quindi $n_1^* \log_2 n_1^* = 2$.
La equazione 6 è ricavata da una relazione tra n_1 e n_2 ricavata da Halstead in [1].

Effettuando le opportune sostituzioni, si riduce il sistema alle seguenti tre equazioni indipendenti:

$$n^* = 2 + n_2^*$$

$$n_2 = n_2^* \log_2 (n_2^*/2)(n_1 - 2) / n^* + n_2^*$$

$$(n^* \log_2 n^*)^2 = \lambda (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 (n_1 + n_2)$$

È possibile vedere come n_1, n_2 non siamo direttamente ricavabili da tale sistema, le cui equazioni devono essere risolte con metodi numerici o grafici.
Halstead fornisce in [1] una serie di curve dalle quali è possibile ricavare graficamente n_1 o n_2 una volta noti λ e n_2^*. Calcolati i valori di n_1 ed n_2, tutte le altre osservabili della software science sono direttamente quantificabili mediante le relazioni già note.

Applicazioni che sfruttano le capacità proiettive della software science sono descritte in [1] e riguardano fra l'altro anche impieghi nell'hardware come il calcolo a priori del numero di piedini in un circuito integrato.

5. CONCLUSIONI

Dopo questa panoramica che ha dovuto essere forzatamente generale ed inevitabilmente incompleta, e che si spera sia comunque riuscita nell'interno di dare almeno un'idea di che cosa sia la software science, riteniamo opportuno tentare qualche considerazione conclusiva.

I tentativi fatti da Halstead di giustificazione teorica di relazioni quali la lunghezza calcolata $\hat{N}$ o lo sforzo E sono inficiati da procementi dimostrativi e passaggi matematici di discutibile coerenza [4].
Nell'implementare le procedure il conteggio degli operatori e operandi n_1, n_2 non sempre è chiaro cosa e come un oggetto che compare in un programma debba essere contato, ossia le definizioni primitive di operatore e operando non sono prive di ambiguità.

I dati raccolti nei molti esperimenti effettuati sono in alcuni casi confortanti, ma non si può certo dire che confermino in ogni circostanza le ipotesi teoriche.

Quello che si può dire è che l'interpretazione operatori/operandi come elementi basici della teoria è probabilmente troppo semplicistica, così come le relazioni quali lunghezza, volume, livello, sforzo, ecc. sono probabilmente solo prime approssimazioni di equazioni più vicine alla realtà in grado di considerare altre variabili indipendenti che influenzano i risultati.

Nonostante tutto ciò, la metrica di Halstead messa a confronto con altre [8], quali il numero ciclomatico di Mc Cabe o LOC (line of code), fornisce risultati con lo stesso ordine di precisione, ciò a dimostrare l'attuale carenza in questo settore di adeguati strumenti di misura.

Altro lavoro deve quindi ancora essere compiuto per poter vantare il possesso di una teoria che fornisca il modello con cui essere in grado di valutare in modo preciso, magari a priori, prima dell'implementazione, l'occupazione di memoria di un programma o il tempo necessario alla codifica o ancora la sua complessità di comprensione. Tuttavia, senza dubbio la software science sembra essere uno tra i più importanti e *completi* tentativi fatti in questo senso fino ad ora.

L'approccio scientistico, basato su alcune originali osservazioni, intrapreso da Halstead costituisce forse la giusta chiave per poter affrontare con successo il complesso problema della interpretazione del processo di programmazione, un'attività fortemente intrisa di fattori umani.

6. BIBLIOGRAFIA

[1] M.H. Halstead, ELEMENTS OF SOFTWARE SCIENCE, New York, Elsevier, 1977

[2] A. Fitzsimmons and T. Love, A REVIEW AND EVALUATION OF SOFTWARE SCIENCE, ACM Computing Surveys, Vol. 10, Mar. 1978

[3] J.L. Elshoff, A REVIEW OF SOFTWARE MEASUREMENTS STUDIES AT GENERAL MOTORS RESEARCH LABORATORIES, in Proc. 2nd Software Life Cycle Management Workshop, IEEE, NY 1978

[4] J.P. Malengé, RESULTATS DE MESURES DES DIFFÉRENTES PROPRIÉTÈS DU LOGICIEL, Iman, Université de Nice

[5] J.H. Elshoff, MEASURING COMMERCIAL PROGRAMS USING HALSTEAD'S CRITERIA, ACM SIGPLAN 11 (5), 1976

[6] M.H. Halstead, ADVANCES IN SOFTWARE SCIENCE, Advances in Computers, Vol. 18 (Yovits, ed.), Academic, New York, 1979

[7] J.L. Lassez, D. Van Der Knijff, J. Shepherd, C. Lassez, A CRITICAL EXAMINATION OF SOFTWARE SCIENCE , The Journal of Systems and Software 2, 105-112, 1981

[8] Vincent Y. Shen, S.D. Conte, H.E. Dunsmore, SOFTWARE SCIENCE REVISITED: A CRITICAL ANALYSIS OF THE THEORY AND ITS

EMPIRICAL SUPPORT, IEEE Transactions on sfotware engineering, Vol. se9, no. 2, Mar. 1983

[9] E. Calvi, LA FISICA DEL SOFTWARE, A.A. 77/78, Tesi di laurea in fisica, Univ. di Milano.

[10] VOCABULARY EFFECTS IN SOFTWARE SCIENCE, IBM Santa Teresa Lab., Tech rep. 03.082 Jan. 1980

[11] C.P. Smith, PRATICAL APPLICATIONS OF SOFTWARE SCIENCE IMB Santa Teresa Lab., Tech. rep. 03067 Jun. 1979

[12] V.Y. Shen and H.E. Dunsmore, ANALYZING COBOL PROGRAMS VIA SOFTWARE SCIENCE, Purdue Univ., Rep. CSD TR-348, Aug. 1980

[13] E. Calvi, D. Marini, UN ESPERIMENTO DI VALIDAZIONE DI UNA EQUAZIONE DELLA FISICA DEL SOFTWARE, Congresso AICA, Bari, ottobre 1979